Position Sensing and Augmented Reality

David John Johnston

A thesis submitted for the degree of

Doctor of Philosophy

at the

Department of Electronic Systems Engineering

University of Essex

December 2001

Abstract

One of the greatest challenges for the emerging discipline of Augmented Reality (AR) is solving the visual registration problem *i.e.* aligning virtual computer graphics accurately over the real world scene to provide the user with a usefully "augmented" interactive experience. To achieve this goal the position and orientation of the user's head must be measured with high accuracy and low latency. There is no general purpose technology that works outdoors or indoors (with unlimited range) to accomplish this.

Addressing the application of reconstructing ancient Roman buildings *in situ* which once existed at a now green field archæological park in Colchester, an AR system has been developed. This consists of an optical look-through stereoscopic VR headset and a wearable computer, with GPS and computer vision being used for position determination. The Visual Positioning System (VPS) finds and identifies specially designed targets within the scene to work out camera location. The targets have unique signatures when the image is turned into a Region Adjacency Graph (RAG) resulting in robustness and reliability (no false positives). The GPS system uses two receivers: one on the archæological tourist and one at a base station in order to perform differential processing for greater positional accuracy.

Initial evaluations were made of the ADXL202 "accelerometer on a chip" and of the LAMBDA method for high accuracy GPS positioning, with a view towards a multi-component hybrid solution for the registration problem. Currently, the VPS can work unassisted indoors or automatically initialise the registration for use of the lower accuracy GPS outdoors. The systems developed are inexpensive: the VPS uses laser-printed targets and a commodity PC for the visual processing, while the GPS is based on two inexpensive Garmin G12 receivers. The software developed in-house has been put into the public domain.

Acknowledgements

I would like to thank the following for their help and encouragement: my supervisor Dr. Adrian Clark; the concurrent long term inhabitants of the VASE lab: Mike Lincoln, Neill Newman and Eddie Moxey; former inhabitants: John Carson, Graeme Sweeney and Christine Clark; and the new members: Panos Ritsos, Andrew Bradley and Alexandros Isaakidis. All contributed to the friendly and helpful atmosphere. My residence in the VASE lab has been a fulfilling experience — a breath of fresh air after staying too long in a "brain dead" research establishment! I am particularly indebted to those who have provided endless technical support without complaining (you know who you are!).

Many Internet contacts have helped me with the GPS work: Sam Storm van Leuwen from Holland; Antonio Tabernero Galan from Spain; Hamish from New Zealand and Chris Hill of Nottingham University.

I am grateful to Dr. Christine Clark and Dr. Philip Crummy who produced the VRML models of the Roman buildings. Two undergraduate students Ipsita Sinha and Aminatta Houma deserve a special mention for their work on the Visual Positioning System (VPS) and the staff at 5D, a special effects firm in London, provided a vital clue (and their code) in order to perform a matrix factorisation I had been struggling with. Juan Bicarregui of the Rutherford Appleton Laboratory helped me with the mathematics of the "target dominoes".

I would like to thank my landlords: Adam Sands and Kim Lewis in Colchester for giving me somewhere to stay during this time, and wish to dedicate this thesis to the memory of my parents: Tom and Vida Johnston who became ill and passed away after my decision to go to Essex University. Despite the two lengthy resulting delays, my supervisor generously kept the offer of a place open.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Virtual and Augmented Realities

Virtual Reality (VR) involves the creation of a self-contained universe within a computer. A user may interact with this environment via computer peripheral technology, subject to certain rules of the synthetic universe. Generally, the universe can be visualised on a computer display and this is the familiar form of existing VR applications, such as games and architectural previewing systems. The universe usually borrows its rules from the physical world, and virtual worlds are often judged by their realism both in "look" and "feel". Of course, VR need not be real in any sense of the word, but the rules should be consistent and analogous (if not identical) to those of the real world to provide intuitive interaction. VR is cited as a way of removing the traditional barriers to human/computer interaction and indeed human/human communication that is mediated by computer.

A major challenge of VR is to find a way of populating these virtual worlds with sufficiently rich scenes, objects and characters. It takes the considerable labour of talented graphic artists and 3D modellers to produce the form of such items; and much effort from programmers, artificial intelligence experts, physicists, spatialised-audio engineers and motion capture specialists to imbue such items with behaviour. Even after all this effort, the nature of interaction remains stylised and the output is a poor relation to the natural world. The effort could be reduced by automated borrowing from the real world, instead of *ab initio* synthesis. However, automated 3D environment and object capture are research issues. A shared virtual world is one way of introducing interesting behaviour, without any AI character development work. Here multiple users each have a presence character or *avatar* in the virtual world, and interaction between the users is what provides the potential for interest. An

archetypal shared virtual world is the virtual chatroom where spatial control is used to reduce the "free-for-all" anarchy (or to replace the artificial mechanisms) of a traditional Internet chatroom.

A second challenge of VR is to provide the appropriate peripherals for compelling interaction. Display devices range from the VR headset (immersive, stereoscopic but bulky and embarrassing), through the VR cave (a high-cost "room" built of back-projection material giving immersive visualisation of up to 4π steradians but stereo only through glasses), to the conventional CRT (no immersion and stereo only with glasses again). Traditional input devices such as the mouse and joystick are limited in their degrees of freedom, whereas custom VR hardware such as datagloves are expensive and tend to be ineffective at worst or difficult to use and support in software at best. In short, there are currently no ideal peripherals for VR. Conventional peripherals are inadequate; more suitable peripherals are expensive but have sufficient drawbacks so there is no clear move to standardisation and a consequent reduction in hardware costs. Until there are sufficient technological breakthroughs, whether evolutionary or revolutionary, VR remains a curiosity, with true VR systems relegated to the small high end market and simulated VR systems being the stock-in-trade of arcade, PC and console-based games.

Augmented Reality (AR) is closely allied to and inherits much from VR, but is qualitatively distinct. AR involves the integration of real and virtual worlds in an interactive system. AR started life as computer graphic overlays for remote video-sensing applications. A virtual tape measure, for example, provides a way of measuring real-distances within the scene shown on a video display. The major challenge for AR is the accurate alignment of the real and virtual worlds in a composite and credible system. The definition of AR is not restricted to the domain of vision, but in practice this is where the majority of the research effort has been expended, and this context will be assumed for all subsequent discussion. To align real and virtual worlds requires knowledge of the position, direction and field of view of the human observer or video camera. The alignment challenge translates into the simply specified problem of determining position and orientation. There is no universal technology that measures position and orientation to the requisite accuracy to support AR in any environment. Certain suitable technologies work in specific environments or contexts but are highly expensive.

AR makes novel demands on the underlying computer system, often adding a high input bandwidth to the high output bandwidth of VR. VR applications are not accustomed to be driven by real-world input, and there is an absence of suitable real-time frameworks for AR applications.

Although AR places additional challenges on top of those of VR, there is a sense in which AR

Figure 1.1: Archæologists at Gosbecks

Figure 1.2: Colchester's Roman wall

applications remove many of the problems of VR. AR only provides a *delta* (*sic*) rather than a complete universe, so the augmentation need not have the full complexity of a VR environment and need only provide a limited service. For example, simply aligning a semi-transparent infra-red view of a scene over the visual equivalent is AR, and this corresponds to extending or augmenting a human's basic senses. It is common for an AR application to have an explicit data-processing or data-retrieval aspect, but this is not necessary.

1.2 Motivation

1.2.1 The Gosbecks Archæological Park

Gosbecks Archæological Park (Figure 1.1) on the outskirts of Colchester is thought to be the home of of Cunobelin, chief of the Catevellauni tribe, who was the most important king in Britain prior to the Roman invasion in AD 43. Shakespeare's eponymous "Cymbeline" is loosely based on Cunobelin. As well as being an important Celtic settlement, Gosbecks later also housed a number of significant Roman buildings: the largest known theatre in Roman Britain; a temple complex; a fort; and a suspected bath-house.

Despite the fact that Colchester (*Camulodunum*) was the Roman capital before London (*Londinium*), a latter-day visitor to the town and particularly Gosbecks will be disappointed if intact Roman buildings are expected. Later buildings have been constructed from the ruins of Roman Colchester, such as the 11th Century Colchester Castle and St. Botolph's Priory but apart from the impressive Roman town walls (Figure 1.2) all that exists are foundations. The only visible indications of Gosbecks's

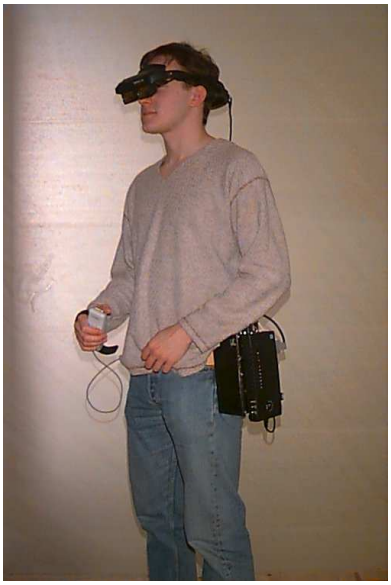

Figure 1.3: Essex University wearable computer

importance are from aerial crop-mark views during dry summers.

1.2.2 Augmented Reality Tour Guide for Gosbecks

English Heritage forbids the on-site use of physical reconstruction at sites such as Gosbecks because of the disturbance to the ground. This prohibition suggests the use of VR technology to recreate the Roman buildings in exactly the **same place** that they used to exist between the 1st and 3rd Centuries AD. As the visitor to the site wanders around, their view of the virtual buildings should change accordingly so that the buildings appear to be located in a fixed position in the real world. The focus of this thesis is the technology necessary to support this AR application *in situ*. The position and direction of view of the visitor must be measured accurately, so that the appropriate virtual scene may be generated and then composited with the corresponding real scene. A mobile VR system is required (Figure 1.3) with a VR headset to provide an immersive stereoscopic experience.

The basic experience of viewing the historic environment can be enhanced through the use of a virtual tour guide (or avatar) to lead the visitor round points of interest with an appropriate commentary. The tour has scope for both interactivity and customisation, according to the interests and profile of the visitor. This is achievable through simple menu selection, which can be presented in audio or visual form. Clearly, the nature of interactions with the computer mediating the tour will be stylised

and capable of being used while on the move, unlike the plethora of menus, tick-boxes, scrollbars and the like beloved of the desktop computer. It would even be possible to place the avatar in period costume and, provided users of the system were networked, to clad other human visitors to the site in virtual period garments. Multimedia could be incorporated in the virtual tour *e.g.* synthetically spatialised audio and relevant video clips.

1.2.3 Technological Implications

The Gosbecks site has a number of special attributes which are of distinct relevance to any AR implementation. Firstly, the archæological park is at the southern edge of modern-day Colchester, with a new low-rise housing estate to the north. The result is that the visible area of the sky forms an almost perfect hemisphere. Secondly, Gosbecks is simply a green field site (Figure 1.1) with extant foundations below ground. Contemporary white line markings delineate the ground plan of the temple complex and the Roman theatre but there is nothing above ground. There is no on-site power at present and though there are plans for a visitor centre this will be subject to strict landscaping.

The freedom of movement in a rural environment necessary for the Gosbecks AR application places a natural dependence on wearable computer technology, and there has been close collaboration with other researchers at Essex University to develop the appropriate platform. Little will be presented of this though the hardware demands of the software will be documented.

The absence of visible remains removes much of the pressure on the AR system to achieve strict registration. Instead, mere spatial stability of registration is required in approximately the right vicinity.

Essentially complete visibility of the sky provides ideal conditions for the use of Global Positioning System (GPS) technology. This is an outdoors navigation system based on a network of satellites. For signals to be received from a satellite, there must be line-of-sight visibility. The more satellites that are visible, the more accurate the GPS position fix. It is ironic that the ruinous nature of the buildings creates ideal conditions for their virtual reconstruction!

1.2.4 Virtual Model Generation

Virtual models may be synthesised either algorithmically by a program or interactively through the use of a 3D-modelling package. Alternatively, a virtual model may be captured from reality, though the technology is far from mature. Ultimately, it should become possible to combine synthesis and

sampling freely. For example, a sampled historically-authentic Roman column could be replicated and scaled to form the columns of a Roman temple otherwise reconstructed by synthesis. Until such captured architectural components models become widely available, how should the temple at Gosbecks be virtually rebuilt?

Much is known about Roman architecture from surviving buildings (see [1]). Roman buildings, certainly those that fulfil civic function, are generally formulaic. That is to say, much of the structure can be inferred from a small amount of information, say, the ground plan and a column width. The rules that underpin Roman architecture were detailed by Vitruvius, a Roman architect and engineer of the first century BC, in his ten book work "De Architectura" [2] which is the only complete architectural work to survive from antiquity. Using the rules of Vitruvius and ground plan information from local archaeologists, it was possible to reconstruct by pure algorithmic synthesis and with considerable confidence virtual versions of various Roman buildings that existed in Colchester. The virtual models used were produced by Dr. Christine Clark and are described in detail in Appendix A.

Where buildings still exist physically, the problem of virtual model generation is the reverse of the AR application *i.e.* the creation of virtual models from 2D views of the real world *cf.* the 2D projection of virtual models controlled by position within the real world. This task of reconstruction is the harder research topic and will not be addressed though others are attempting to do so. However, the difficulty of creating virtual worlds of sufficient richness to be interesting should be noted. Many agree that VR has failed to live up to its early hype, and the author's view is that VR will only take off once the "software titles" become available, and that this software can only emerge from effective reality capture systems. This thesis is concerned with the intuition and effectiveness of replaying VR models. Note that models being used are effective for prototyping the system, and were not designed for photo-realistic rendering. Mastery over the latter domain has been well demonstrated by the special effects industry. Entire historic cities have been recreated in virtual form, not just by Hollywood but by archaeologists and architects — so the availability of virtual models has been taken as a given.

1.3 Contributions from this Work

The work presented in this dissertation addresses the major challenge of AR, namely to align real and virtual worlds. The practical work has investigated a number of different positioning technologies. To close the loop a software framework was constructed which incorporates these positional devices and

supports AR applications. The particular contributions from this work are:

Assessment of Differential GPS for outdoors AR. To obtain greater GPS positional accuracy (such as is required for AR) additional receivers can be placed at fixed locations nearby. This is called differential GPS (see Chapter 3). To assess differential GPS as a position-sensing technology for outdoor AR, in environments such as Gosbecks, necessitated the production of real-time differential GPS software. The results were analysed to guide iterative refinement of the processing techniques employed.

Differential GPS systems are generally bespoke, closed-source and expensive. This work allows two cheap GPS receivers to form a differential system and the software developed has been placed in the public domain. The software runs on a PC and allows the GPS receivers to be connected to the RS232 port of any machine on the same network for full flexibility. The Internet has the potential to support differential GPS and supply high accuracy positioning inexpensively on a global basis but this is almost totally underexploited; various novel schemas are proposed.

Production of a real-time computer vision positioning system. The vision based system was developed as a support technology to maintain correct visual registration for outdoor AR. Despite its occasional operation (at set-up and thence periodically) it was identified as a key requirement. Software was written because no such public domain code existed to assist with AR registration. Conditions can be controlled indoors so the vision system can work unassisted.

The system is based on specially-designed targets and software to detect and identify them. It is a particularly low cost system, requiring a PC of only modest processing power; a video grabbing board; a small commodity video camera; and a laser printer to create the targets. As well as measuring the position of the camera (fixed targets, mobile camera), the system can be operated to measure the position of objects with attached targets (mobile targets, fixed camera). Using this capability, a 3D “video joystick” was developed. By manipulating a small cardboard cube with a target on each face in the field of view of a video camera, full six degree-of-freedom information can be fed into an application. The user can then “directly” manipulate virtual objects on-screen, without the cumbersome mode changes forced by a conventional two coordinate valuator such as a mouse or joystick.

Production of an AR visualisation framework. This takes the position and orientation information from the positioning technology and produces a stereoscopic computer graphic rendition of a virtual world that aligns appropriately with the real one. This software allows the suitability of the various position determining technologies to be verified (or not) in practice. The software framework is flexible in that it can work without a full 3D positioning technology being present: position and orientation can each be derived independently from several devices — even a conventional peripheral such as a mouse may be used.

Evaluation of inertial sensor technology for AR. The author's view is that the best current technology for outdoors AR (the focus of this thesis) is a combination of high-accuracy GPS and inertial technologies, as found on aircraft. How useful are ultra-cheap micro-machined "accelerometers-on-a-chip" such as the ADXL202 two axis accelerometer from Analogue Devices to fulfil the rôle of an inertial sensor for AR? Support software was written for the ADXL202 in order to produce results, and these results were analysed for suitability.

Overall, the work may be summarised as cross-disciplinary, trying to capture a number of different technologies at low cost and without specialised hardware to solve the problem of position determination for outdoors AR, identified as a research issue by Azuma [3].

Clearly the applications of the system that has been developed go far beyond Gosbecks. For example, it could be used at any historic site for recreation; to add virtual labels to exhibits in a museum; or to provide a virtual presence experience in either an imaginary virtual world or a virtual world that has been captured from reality. The component technologies that have been developed individually have utility outside the application domain, and these are described in greater detail within the appropriate chapters of this thesis.

1.4 Structure of this Thesis

Chapter 2 presents a literature review that explores the existing art in the area of Augmented Reality: in particular examining positioning measuring technologies, which were identified as the main focus for this work.

Chapter 3 explains the operation of the GPS system and derives the mathematics behind a position fix calculation. The system is a complex one, and it is only through an understanding of the various components that the error sources and techniques to minimise these may be appreciated.

Chapter 4 describes the “homebrew” approaches taken to develop differential GPS systems using inexpensive GPS receivers. Off-the-shelf systems of suitable accuracy were prohibitively expensive, and so were not used. Chapter 5 deals with the aspects of target design and target recognition in a visual positioning system. Even using GPS technology, it is necessary to initially register the real and the virtual in any AR application. A vision based system is ideal for this, and indeed ideal for a self-contained indoors AR system. An in-house position measuring system is developed based on the use of a video camera and printed targets. The system is called the VPS (Visual Positioning System).

Chapter 6 describes the geometric number-crunching done by the VPS to determine position once the targets within a video frame have been identified. Chapter 7 examines the scope for using multiple position determining technologies to create a hybrid virtual sensor with superior qualities. Inertial Measurement Unit (IMU) technology is considered. Chapter 8 develops a suitable software architecture for AR visualisation and describes in-the-field testing of the resulting system and target hardware. Chapter 9 presents significant and overall conclusions from the project, presented in a self-contained manner. This includes a summary of future work that has been identified.

Chapter 2

Augmented Reality - Systems and Technology

2.1 Introduction

The project to recreate *in situ* buildings that existed 2,000 years ago at Gosbecks Archæological Park in Colchester draws on the fields of Augmented Reality (AR) and wearable computers. There has been much research in AR (Azuma's survey [3] is definitive) and there are already wearable computers in the marketplace such as the ViA II [4]. However, there has been little work on the obvious synergy between the two in replacing the desktop metaphor with something more appropriate for mobile computing. Despite the recent emergence of the first encompassing textbook [5], integration is a still a focus for research rather than a reality.

This chapter examines the state-of-the-art in AR and research that is relevant for the usability of mobile applications. Firstly, the problems that make implementing the Gosbecks application a considerable challenge are described (Section 2.2). The parent subject areas are discussed briefly: wearable computers and AR (Sections 2.3 and 2.4). Then, the literature is broken down into the three categories: interaction, systems integration and technology (Sections 2.5, 2.6 and 2.7 respectively). Note these are presented in a top-down sequence to aid understanding. Only once application demands and some aspects of human factors have been discussed does it make sense to discuss system architecture and the details of particular technologies.

2.2 The Challenge

General platforms for position-mediated information processing do not exist. Indeed, there is much debate even regarding the form that a wearable computer might assume. To construct an AR platform, let alone develop an application for such a platform, there are three not insignificant hurdles to overcome:

1. Technological. There is no established, or even suitable, technology for general position determination with the range or accuracy necessary for the use of AR techniques on a wearable.

2. Systems Integration. There is no general framework for integrating position determining devices and output devices suitable for AR with standard delivery systems for Virtual Reality (VR). *Ad hoc* AR software platforms exist, such as the in-house one developed at the University of South Australia called TINMITH2, which supports the architectural AR system described in [6].

3. Mechanisms for Interaction. It is yet to be established how users can best interact with wearables. Research into AR technology as a mediating agent for interaction has been little investigated due to the barriers presented by 1 and 2. Wearable applications are novel, and it is likely that the nature of the interaction they demand will be novel too. One of the few relevant developments has been undertaken at Columbia University. The system, called MARS [7] (Mobile Augmented Reality Systems), presents AR interfaces for both a roaming outdoor user and remote indoor experts. This heterogeneous computing environment shares the same repository of campus related information allowing the latter users to guide the former.

The research for this thesis, through the development and the evaluation of a novel prototype application, will touch on all three challenges above. To address either 2 or 3 fully is overly ambitious in the context of an individual Ph.D.. Consequently, the more modest plan is to construct a flexible software framework for human-wearable interaction that will permit sufficient initial investigations into both fields to produce a working system. However, this same framework will enable future more detailed research to be carried out at Essex University and elsewhere.

A design goal is to use both standards and off-the-shelf hardware and software components where appropriate, to demonstrate the accessibility of AR technology. An approach that builds on existing practice, and provides evolutionary routes from current systems to a convergent target architecture is more likely to be successful.

2.3 Wearable Computers

The definition of a wearable is as straightforward as a computer that can be worn. Various esoteric forms of wearable have been suggested [4] but, in the context of this research, we take a wearable to be a fully-fledged computer at least the equivalent in power of a high-end desktop machine capable of running industry standard software. The exact nature of the specialised peripherals that one might expect in a future commodity wearable is uncertain at present, but for this project they will involve at a minimum some form of input location device and a head-mounted see-through display.

Wireless networking is possible using cell-phone or radio-LAN technology, though this is less critical for applications which do not require dynamically updated information. There has been much debate whether a wearable will be more than just a "PC in a rucksack". For example, it may consist instead of a number of compute nodes distributed about the body which can inter-communicate as well as connect to services and peripherals in the outside world via short range wireless. However, this research is largely independent of the eventual form of a wearable, and is concerned primarily with issues of usability.

The omnipresence of wearables heralds a new form of communications network where the nodes are people. This is known as a "piconet" [8]. Within a piconet, communication sub-networks are continually being dynamically created and destroyed, as individuals move in and out of radio-communication range of each other. Bluetooth [9] is an emerging relevant standard for short range, high bandwidth, radio frequency digital communication.

A piconet is capable of holding much information in a similar manner to Usenet. In particular, it is especially useful to locate individuals in time and space in much the same way as an active badge system [8]. For example, fixed nodes could donate absolute position information to the system — this could usefully locate individuals within a few node hops. The richness of the information available depends on the density and nature of the sensors placed on the nodes. We can only begin to guess at the capabilities of such a system. The software challenges of coordinating such a network are considerable. It is worth viewing a multi-computer wearable as a "personal piconet".

2.4 Augmented Reality

Augmented Reality is the combination of virtual objects with reality in an interactive system. "Mixed Reality" is the less common term that has come to define the general interactive combination of the

virtual and the real. AR was initially developed as a visualisation technique but has subsequently developed into a research topic in its own right. [10] presents a good summary of the distinctions between AR and the more familiar VR.

VR requires much output bandwidth (naturally limited to forms that the five human senses can detect) but little input bandwidth; AR creates the additional demand of high input bandwidth but with no restriction on form. This suggests the intriguing use of AR to amplify human sensory capability, where raw and processed data from IR and ultra-sound sensors, say (which humans do not possess), could form the superimposed information.

Taxonomies have been developed to categorise mixed reality systems. A well-considered schema developed in [11] is based upon a three dimensional categorisation:

Extent of world knowledge

How much knowledge does the computer have of the real-world?

Reproduction fidelity

This is a consequence of rendering and output device quality *e.g.* perceived image quality.

Extent of presence metaphor

To what extent does the observer feels "present" in the displayed scene?

For our type of AR system, the presence metaphor is total as a real-world presence is the starting-point of the system. The reproduction fidelity is not vital: as long as the image quality is "good enough" questions of usability can be investigated. On the other hand, "extent of world knowledge" is the unknown axis of our system that has to be investigated to determine usability, and of course the cut-off point is variable depending on the effort available and the required sophistication of the final product. How complete a model of the real world does the computer system need? This may range through the following:

Nothing Virtual graphics may be overlaid on the real world, yet present an independent (and unrelated) information channel.

Positional The virtual objects are positioned and aligned in real space.

Static model based Static real-world objects are fully modelled so virtual objects can be occluded by them and real/virtual object collision can be detected. Details of real world objects are generally

supplied at start-of-day, but may be dynamically inferred or measured, possibly by computer vision techniques.

Dynamic model based Identical to the static case, except that the system is able to cope with movement of real world objects. Clearly a system capable of tracking a number of objects in a scene simultaneously is either demanding in terms of computer vision capability or tracking hardware. The system described in [12] which computes interactions between virtual and real objects in video sequences represents the state-of-the-art in "extent of world knowledge". Note, however, the constraints of such a system: the camera's trajectory is known (being controlled by computer) and the computations are performed off-line.

An overly simplistic but nonetheless revealing one dimensional schema is "The Virtuality Continuum" on which any mixed reality system may be placed. It ranges from Reality at one end to Virtual Reality on the other [11]. The author proposes a more revealing categorisation which assigns a Sample/Model directed graph to each mixed reality system representing the flow of information within the system. Some examples are presented in Figure 2.1.

An M node represents data held within a mathematically perfect model An S node represents data which has been sampled from the real world or from such a model. Generally an arc $S \rightarrow M$ represents a interpretive process: for example the equation of a straight edge could be inferred from a digital photograph of the horizon. However, $S \rightarrow M$ could also represent the move from raw sampled image data to a mathematical model of an image containing this sampled data. An arc $M \rightarrow S$ represents a rasterisation process. D represents the location of the display in the system, and FW represents the functional real-world data *i.e.* it acts as real world data, but could actually be computer generated. This generalisation allows graph descriptions to be composited.

This scheme avoids the thorny issue of assigning the labels "real" or "computer generated" to information sources. For example, is a sampled and reconstructed model of a human being real or artificial? It is a moot point, especially when it may look like someone who actually exists but is artificially animated entirely by computer.

This network characterisation will capture the nature of the graphical display, but will not give any clue as to the extent of world knowledge linking the various components. Note that "model mixing" is easily expressible in this graph format, yet generally escapes categorisations in AR discussions. Model mixing is the appropriate technology for virtual movies, that contain avatars representing known actors.

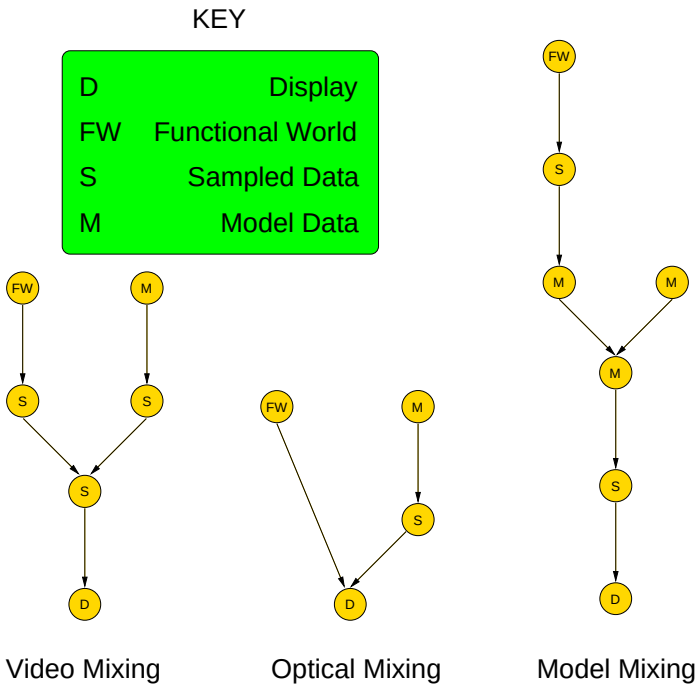

Figure 2.1: Examples of Augmented Reality systems

2.5 Mechanisms for Interaction

There is a vast gulf between the appalling interactive interfaces presented by the majority of computer applications (where a metaphor of any kind is absent) and the frankly off-the-wall metaphors behind the systems in the toy-cupboard of such institutions as MIT [4]. The difficulty is combining effective new metaphors with useful production applications and being able to market the results.

A series of Tangible User Interfaces (TUIs) developed at MIT and elsewhere is presented in [13], which take advantage of the natural way humans interact with physical objects. Virtual objects can become associated with physical objects and moving the physical object will move the virtual object as well. Some genuinely valuable system have been produced, but for widespread use there will have to be some form of standardisation upon Physical Icon ("Phicon") technology. The suspicion is that, even if such systems do arrive, this will be a long time in the future. Perhaps domestic products may be the route by which this technology arrives? A particularly appealing product is the Marble Answering Machine, where each message is represented as a marble. The marble can be placed in an indentation

in order to be played or placed on an augmented telephone thereby dialing the original caller. The concept behind the Phicon is one of physical embodiment.

The “virtual Phicon” may be the emergent interactive metaphor of AR. Here, the virtual control object has many of the properties of a real physical object, but does not have to be physically instantiated with the concomitant physical tracking problems. The virtual Phicon can be controlled by any input device with a sufficient number of degrees of freedom through a relatively intuitive mapping. This mapping is dynamic, as the input device can become associated with a number of different virtual Phicons. An on-screen cursor is nothing but a virtual Phicon controlled by a mouse. So it can be seen that virtual Phicon control is merely a generalisation of existing practice.

It is difficult to introduce new physical input devices: the desktop metaphor presented through a variety of windowing systems has settled, presumably irrevocably, onto the keyboard and mouse. AR not only makes the desktop metaphor redundant but demands new technology, so it is in the long term interests of the area to elect effective and appropriate input device technologies as early as possible.

2.5.1 Positional Mediation

Position mediation takes a number of forms: context and coordinate driven. A museum’s information system may start up on a wearable when the context is determined to be “in the museum”. A potted biography of another individual may pop up when in the presence of that individual. Both of these are “context driven”. Alternatively, the wearer’s view of a virtual object may change as he or she walks around it, based on his or her position in space. This is “coordinate driven”. In fact, context may actually be derived from an interpretation of coordinates or may indeed be based on a technology completely unrelated to position determination.

Aside from these fairly transparent uses of positional mediation, it is an open question whether a useful interactive dialogue with a machine (in essence a language) can be based solely or indeed even partially upon the mechanisms of position mediation open to a wearable user. In the opinion of the author, the major leverage of a wearable (aside from the omnipresence afforded by its portability and the accessibility afforded by its peripheralisation) is positional mediation.

There are three ways to derive context:

Model-Based The system uses a positioning technology and a database to infer the context.

Inference-Based The system derives the context from interpreting the universe around it (say by

computer vision). Coordinate information is not used.

Direct The system is told the context (*e.g.* presence within a particular room) by a context technology.

There are two ways to determine position:

Inference-Based The system derives its position by interpreting the universe around it (say by computer vision). Some computer model must pre-exist. In a vision based inference system, an important design decision is whether to rely upon *fiducials*[1], easily identifiable markers which may be placed about the scene. Clearly, this simplifies the problem, but restricts the generality of the solution. Other than for development systems; indoor systems of limited range; and for initialisation the author is not in favour of fiducials, as these are counter to the philosophy of mobile AR — especially outdoors.

Direct Direct use of a positioning technology.

Figure 2.2 summarises the interconnections. Different techniques are appropriate in different circumstances. In some cases a model of the area must pre-exist; in other cases the area must be instrumented. Note that instrumentation for direct context technologies may be independent of any instrumentation necessary for direct position determination.

Though some interpretation is in theory still possible where neither a model nor a direct positioning technology exists, we do not address this realm as large errors are likely. Out of the reach of maritime radio beacons (instrumented space for direct positioning), sailors may still navigate by the stars (model based), but navigation in fog would be risky!

In fact a strict distinction cannot be drawn between measured and interpreted position. With GPS, for example, the algorithms used to gain high accuracy are fallible — and perhaps level of confidence is the best way of looking at such systems. Indeed, even a model may be regarded as a way of instrumenting space.

2.5.2 Position Determination

Critical factors in position determination are the technical issues of range and accuracy. For example, GPS technology only works outdoors or within a short distance from a window, whereas more accurate magnetic positioning systems work indoors but have ranges of only a few metres. The limited range

[1] *fiducia* is the Latin for trust

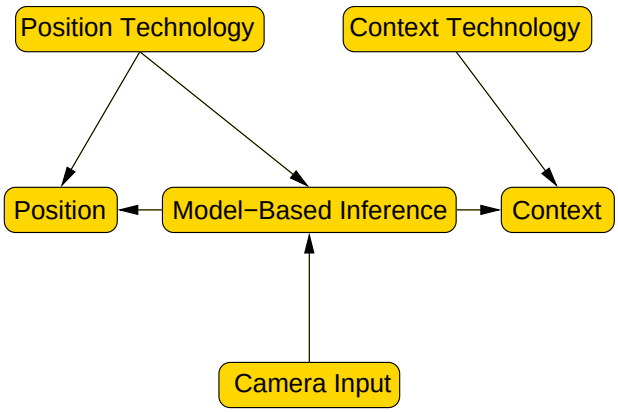

Figure 2.2: Context and position determination interconnections

of most positioning technologies is at odds with the complete freedom of movement demanded by a wearable user.

Quantitative positioning specifications qualitatively affect application usability. For example, to overlay a virtual object in the field of view of a user one metre out of position may be acceptable if that object is a virtual building in the distance but this would not be acceptable if the object were supposed to be a virtual human at a typical distance for social interaction. The author's target positional accuracy for the Gosbecks application is 10 cm, which corresponds to the typical clearance of a human when passing through a doorway. All positional technologies have been evaluated with this figure in mind. The AR user then has a fighting chance of interacting with virtual architecture at the most basic level.

2.5.3 View Determination

To present a correct stereoscopic view of a virtual object requires more than just the relative position of the viewer. The system also needs to know the direction of gaze (yaw, pitch and roll angles); the

field of view; the inter-pupillary distance; the accommodation (focus and depth of field of the eyes); and the vergence (or angular disparity) of the eyes. The technology to measure the latter two quantities does exist but is esoteric and expensive. In short, it is currently impractical to fully mimic all possible depth cues of reality, but any architecture developed must be capable of carrying this information for future compatibility. Note that it is difficult to isolate view determination from technology. For example, with a holographic display system many of the parameters above would not be necessary.

To correctly render a virtual scene that is to be overlayed on reality, requires matching the existing lighting conditions. Real shadows must fall on virtual objects and virtual shadows must fall on real objects. Systems exist which do this [12] but these involve detailed modelling of the real world from the analysis of video sequences.

Full-fidelity AR is overwhelming in its complexity. Instead, we merely aim to establish sufficient realism to provide usable rather than cosmetic returns. Depth cues are many. The following table details these by category based on those of [14], with an indication of whether each will be supported. A question mark indicates where a particular depth cue could be supported with little extra effort, but is not planned.

Pictorial Depth Cues		
Name	*Description*	*Supported*
interposition	nearer objects hide distant ones	NO
linear perspective	parallel lines converge	YES
texture perspective	texture density increases with distance	YES
atmospheric perspective	loss of detail and colouration with distance	?
relative brightness	illumination gives inverse square information	?
shadows	self-explanatory	?
Kinetic Depth Cues		
Name	*Description*	*Supported*
relative motion parallax	distant objects stay still, nearer objects move opposite to direction of motion	YES
motion perspective	nearer objects appear to move faster	YES
kinetic depth effect	rotating objects betray their 3D structure	YES
Physiological Depth Cues		
Name	*Description*	*Supported*
convergence	rotation of eyes to fixate upon object	NO
accommodation	change in focal length of eye	NO
blur	self-explanatory	NO
Miscellaneous Depth Cues		
Name	*Description*	*Supported*
binocular disparity	situation complex as this varies with convergence	YES
perceptive factors	biases associated with expectation, *etc.*	N/A

Research described in [15] indicates that simple binocular view disparity stereoscopy provides simple and effective depth cues, that can be rapidly processed by the brain. Many of the results from teleoperation are relevant for AR. Indeed initial AR systems consisted of computer graphics overlays to enhance teleoperation [16].

2.6 Systems Integration

Given that most of the hardware to be used can simply be physically connected together, the majority of the systems integration will involve software effort. A small amount of hardware integration was both carried out and is proposed to create AR sensors of the desired capability. What follows is a brief discussion of the proposed data formats, software components and languages for the integration exercise. Naturally these comments will be tempered in the light of experience (see especially Chapter 8 on visualisation) but it is useful to present the "nominal" software technologies that in theory are suitable.

2.6.1 VRML2

The second version of the Virtual Reality Modelling Language (VRML2) [17] has been standardised and is the strongest contender for a network transparent representation of virtual worlds. While it reads as a rather verbose scene description scripting language, VRML2 can represent more than just static data. Through "events" VRML2 enables animation and autonomous objects (or "nodes") with behaviours that can be coded in "scripts" written in Java or JavaScript. This allows for program control of a VRML world. Many VRML-informed Web browsers exist. Although VRML is equally appropriate for AR, most VRML exists in the traditional VR arena. Most AR systems are *ad hoc*. Our approach is to use standards such as VRML through the development of appropriate interfaces and frameworks.

2.6.2 Java

Java [18] is a much heralded object-oriented, network-transparent, platform-independent programming language which is claimed to be appropriate for Web use. It was initially the language of choice for this project, though where low level device interfacing or high levels of performance are required it is acknowledged that alternative languages may also be employed.

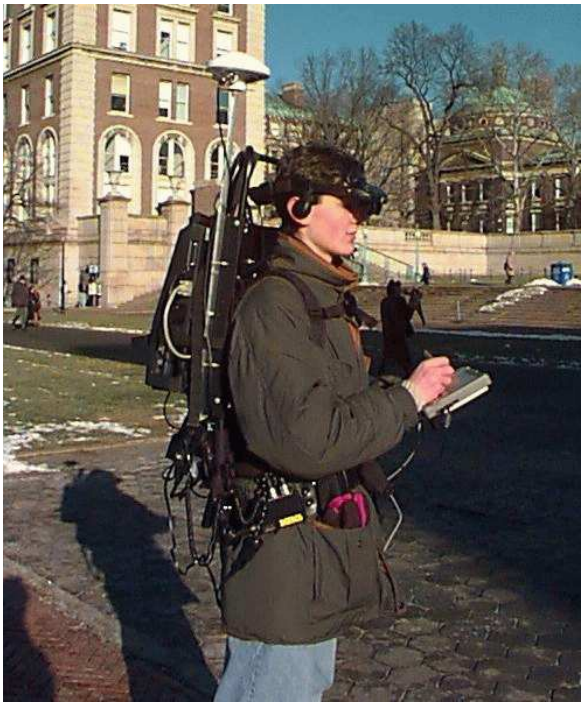

Figure 2.3: The Touring Machine: Columbia University

2.6.3 Positional System Issues

It may appear that a great deal of attention has been paid to the position determining subsystem within the project. This is because the appropriate technology does not currently exist, and it was anticipated that much of the effort in developing the system would have to be expended in this area. This conclusion has been reached independently by others looking into the field such as Azuma [3], who revealingly also states: “Tracking an AR system outdoors in real-time with the required accuracy has not been demonstrated and remains an open problem”. The “Touring Machine” shown in Figure 2.3 is one of the few examples of an outdoors AR system. However, the application is not that demanding of strict registration because it merely annotates buildings on the campus of Columbus University with labels. Problems with the operation of differential GPS in urban areas are reported, and the positional accuracy is only around 1 metre.

Indoors, no off-the-shelf position determining system has enough range for a wearable. Outdoors, though GPS receivers exist with suitable accuracy, latency and update rates; they are currently too expensive, bulky and heavy to be part of our wearable configuration.

It is possible that a sensor combining GPS, inertial and even computer vision components may

provide the best benefit/cost ratio, and there is a considerable challenge in developing sensor fusion software that creates a virtual position determining device with properties masking the peculiar deficiencies of each of the component technologies. This presents an interesting mathematical problem: given a number of different measurement streams of a particular quantity, how do we combine these at any instant in time to give the most reliable measure? Each stream will have an associated noise, accuracy, frequency, and drift. Solving this general theoretical problem will undoubtedly give great leverage to the whole area of hybrid sensors. Of course, the ideal situation is being able to place the application defined characteristics of the hybrid sensor that is required as the input to a black box. The output of the black box will be the identified component technologies required and the algorithm by which their individual measurements are combined.

From a systems point of view, a clear interface can be identified at this virtual hybrid device layer. [19] identifies a middleware layer in a sensor-driven architecture which turns the raw sensor data into services, that will be used by higher level applications. High performance sensor technology is critical to a wearable, and the necessary responsiveness of the system suggests a real-time architecture, and one that is potentially distributed where compute nodes are dedicated to particularly demanding tasks. However, a generic real-time software architecture for AR applications is simply not available let alone discussed in any detail in the literature. Clearly, a conventional operating system underpinning AR is hardly suitable except as a development platform.

2.7 Technology

2.7.1 Augmented Reality Display Technologies

There are numerous discussions on the variety of display technologies appropriate for AR [20], [11]. It is perhaps worthwhile to describe the technologies peculiar to AR, as these are less familiar than those of Virtual Reality which may be adapted for AR use.

Optical See-through Head Mounted Display (HMD) A head mounted display which merges virtual and real worlds optically by using a half-silvered mirror.

Video See-through HMD Virtual images are overlayed onto video images of the real scene. There are some advantages to this approach as correct registration of the two images has already been achieved. Occlusion between real and virtual objects is easier, and computer vision techniques

are more easily accessible as the source data already exists. The latency in both real and virtual paths can be more easily controlled to tally, but a reduced resolution "reality" has many obvious disadvantages for the wearer.

Video See-through Head-Up Display (HUD) This partially covers the viewer's dominant eye so that a virtual floating window is presented in front of the viewer's face. It has been found that an image presented to a single eye is minimally intrusive, though depth information through stereoscopy in the virtual scene is lost.

Video See-through Palmtop This is a less common type of system described in [20] which uses the magnifying glass metaphor and a palmtop computer to merge virtual images with real ones within the window of the palmtop's display. This is less invasive than a head-mounted display and it is claimed to have performance benefits and a more favourable user reaction. The palmtop contains a miniature gyro sensor to determine the orientation of the device and a vision-based ID recognition system to detect a rough position of the device in the real world and of objects placed in front of it.

One major technical difficultly for head-up displays is presenting a wide field of view matching that of the human visual system, and the specification of such displays will generally include the angle the image subtends on the retina. Cave technologies [21] (where the internal walls of a small room are used as a multi-screen projection surface) remove this display restriction, but they are only suitable for pure VR (rather than AR) applications.

2.7.2 Direct Context Technologies

These are generally based on an occupancy model. Is object O_1 in the vicinity of object O_2 or not? Generally the answer is binary, but sometimes the signal quality can be also detected and by viewing a network of detectors as a system, additional information can be inferred. For example, an object being registered by two detectors must be within the range of both.

Locust

These are small cheap electronic devices developed at MIT which broadcast and receive IR pulses. A wireless network of locusts [22] in known positions can be constructed to span any given domain

within a building. It is possible to interrogate the location of any individual, provided that individual is themselves wearing a locust.

The granularity of localisation is a room or zone of a room, depending on the density of locusts fitted. Accurate positional information is not available, but basic contextual information is available at a low price.

Passive RF Inductive Loops

This Radio Frequency IDentifier (RFID) technology [23], is the radio frequency equivalent of bar-codes. An installation consists of a number of passive inductive tags and a RF broadcasting/receiving system. Within a certain range (up to a maximum of 25 cm) a passive tag will be able to derive enough power from the broadcast signal to transmit back its unique ID and other information. The tags can each be reprogrammed with a new ID and other information, by a similar transmission. Unlike bar-codes, line of sight is not required and the tags can be invisibly sewn into lapels, clothing, *etc.* . Given the limited range, the RF base stations have to be located at doors, *etc.* in order to track individuals through their room-transitions. Essentially, the functionality is the same as the locust device.

Bluetooth

Bluetooth [9] is an emerging standard for short range radio-frequency communication. The intended range is 5-10m, but if pushed the technology could extend to 100m. Though more a digital communications technology, the limited range enables it to be used as a context technology.

2.7.3 Position Determination Technologies

We are looking specifically at human movement tracking systems. These can be classified [24] as inside-in, inside-out or outside-in, depending on the relative positions of sources and sensors.

Inside-in Sensors and sources both on the body *e.g.* a glove that detects finger flex. Inside-in systems are suitable for relative tracking of small body parts and have small form factors and unlimited range.

Inside-out Sensors on the body and external sources *e.g.* a camera-based head-tracker using a ceiling as a reference. Inside-out systems provide absolute position, but the range and accuracy is

limited by the need for an external source. The medium form factor makes these suitable for attaching to medium to large body parts.

Outside-in Sources or markers on the body and external sensors *e.g.* a camera system that tracks reflective markers on the body. Outside-in systems suffer from occlusion and a limited workspace but are regarded as the least obtrusive. Tracking of smaller body parts, which are more prone to occlusion, must be carried out by many cameras or by restricting the workspace.

For useful wearable interaction, how many body parts require to be tracked simultaneously? Our initial objective is the minimum tracking necessary to drive an AR display, which consists of the head position and direction of view. There are detailed surveys of (single) position determining systems [24], [25], [26] and related sensor technologies [27], but only those relevant to a wearable system will be addressed. Clearly an inside-out technology is most appropriate for a wearable. To cover any significant range, an outside-in system would require multiple external static sensors increasing cost, or a single external mobile sensor increasing complexity. In the latter case, the problem is literally moved elsewhere. How does the mobile sensor know where it is?

The problematically demanding tracking requirements for AR are presented in succinct form in [28]. It is suggested that to maintain a reasonable degree of registration between the virtual and the real, an angular accuracy of 0.5° and a total system latency of under 2 ms are required. These are difficult figures to achieve in practice, but represent a usefully documented baseline. The navigation system described in [29] claims to be usable, though an optical see-through display is used in conjunction with a GPS positioning system which gives an accuracy of "only" 1m. Dissertation [27] demonstrates that predicting head movements especially if inertial sensors are used is an effective way to reduce dynamic registration errors which are usually larger than the static ones. Accuracy is quoted at 5 to 10 times greater when prediction is used. This suggests that intelligent software and the use of Kalman filters may be able to reduce the effect of intrinsic latency within systems.

Mechanically-Based

A mechanically-based position measuring system requiring a linkage between the moving object and the stationary environment is definitely unsuitable for a free ranging wearable system.

Figure 2.4: HiBall Tracker: University of North Carolina

Vision-Based

Techniques are based on triangulation in the way that many range finders operate. The state of the art is represented by something such as the HiBall Tracker from UNC [30] [31] shown in Figure 2.4 . The system, which is now a commercial product, uses ceiling panels housing LEDs, and a miniature head-mounted camera cluster called the HiBall. This specification is presented in the follow table.

UNC HiBall Tracker Specification	
update rate	$>$ 1500 Hz
latency	$<$ 1mS
position noise	$<$ 0.2 mm
orientation noise	$< 0.03^{\circ}$

Scaling the range is as simple as installing more special ceiling tiles and clearly this is not a system that would work outdoors. The positions of the LEDs in the ceiling do not have to accurately surveyed, as the system autocorrelates. The group's next goal is to use image-based and inertial

tracking to enable the system to work in non-structured environments.

UNC has been involved with tracking for 20 years, and the HiBall has been under continual refinement and study: the sources of errors have been categorised and modelled. The major player in AR tracking, Azuma, has moved from UNC to HRL Laboratories and continues to be at the cutting edge: looking at combining inertial and vision-based tracking [32] [33], electronic compass technology [34] and in particular the combination of gyroscopes, tilt sensors and electronic compasses [35]. It is to be noted that within this context Azuma is only looking at orientation from a fixed outdoors location. He calls this a "base system" [36], and in working with just these 3 Degrees Of Freedom (DOF), he is acknowledging the difficulty and unsolved challenge of measuring a full 6 DOF outdoors. While many AR researchers refer to GPS as a potential solution, it is clear that the technology is being paid lip service and few have examined the techniques for high accuracy in any detail. Azuma refers to research at the Rockwell Science Centre to determine orientation by working optically with a silhouette of the horizon, so even a solution for a base system has not been established.

Wave Propagation

There are four basic techniques:

Pulse timing: measures round trip for signal to bounce off a reflective surface

Phase comparison: a carrier wave may be modulated at different wavelengths. Phase difference between transmitted and received signal can give distance modulo the wavelength to a fraction of the wavelength.

Doppler methods: the frequency shift gives relative velocity which in combination with other known factors can give position.

Interferometry: used to measure distance in the same way as the original Michelson Interferometer.

Radio

Many radio-frequency location technologies exist, such as Radar which is based on pulse timing. However, the only off-the-shelf RF system suitable as a personal technology is GPS (Global Positioning System). This is is based on a network of satellites that encircle that earth.

Much conflicting information on the positional accuracy of GPS technology exists. The greatest problem is that accuracy is dependent on many factors such as the number and layout of GPS satellites

within line of sight. Using information from a second GPS receiver, in a technique called differential GPS, gives more accuracy, than say the upgrade from the use of deliberately degraded but unencrypted civilian GPS signals to non-degraded but encrypted military ones. We are concerned with performance in a practical AR context: one using simple civilian GPS in a real-time (not time-averaged or post-processed) manner. With differential GPS, the closer the second receiver is positioned the more accurate the differential correction that can be applied, so figures for baseline distances of less than 30km and more than 30 km are both presented in the nominal table of achievable accuracies below. Note there figures are correct prior to 1st May 2000, when civilian degradation was switched off.

Technology	Accuracy	Cost (1994)
Basic GPS	100 m	$1000
Differential Code GPS ($>$ 30km)	10 m	$5000
Differential Code GPS ($<$ 30km)	1 m	$5000
Differential L1 Carrier GPS	0.1 m	$10000
Differential L1/L2 Carrier GPS	0.01 m	$10000

There are two forms of differential GPS: one using the information content of the signal (differential code) and the other measuring the phase difference of the carrier wave (differential carrier). The ultra accurate differential carrier L1/L2 systems extract phase information from both frequencies (L1 = 1575.42 MHz and L2 = 1227.60 MHz) broadcast by the satellites.

Many relatively inexpensive units have the capability of using a differential "correction" signal to improve accuracy. The most common way the differential signal is supplied is via a series of maritime radio beacons situated round the coast. Less often two receivers at a suitable distance are used, which is sometimes described as "Poor Man's Differential GPS".

Paper [37] describes a location system based on GPS chip sets. By using pseudolites (fixed low-powered transmitters or "pseudo-satellites" broadcasting similar information to GPS satellites [38]) it is claimed location accuracies of a few millimetres can be obtained in volumes ranging from rooms to football fields, using a tetrahedral arrangement of only four pseudolites. This technology could bring GPS indoors and into urban areas. Over the lifetime of this thesis, pseudolites have moved from a concept to hardware realisation for experimental research. This is encouraging given that the specification beats all existing indoor position technologies for accuracy and range. Radio waves have the ability to penetrate objects better than sound, IR or light so pseudolites are worth watching as an emergent occlusion-resistant technology. Of course, the frequency bands would have to be checked

for government restriction and/or licensing. However, in the US, transmitters with a power of less than a watt (sufficient for typical pseudolite operation) are excluded from legislation.

Acoustic

Ultrasound gives room-size range and millimetre accuracy, but the receiver must roughly be pointing in the direction of the transmitter and thereby hindering movement. Systems are available from Logitech. Acoustic systems are vulnerable to noise, and due to variations in ambient temperature they are difficult to make accurate at longer ranges. The disadvantages have caused the technology to be regarded as rather passé in AR circles.

Laser

Laser ranging devices exist, though laser scanning and vision technologies are more often combined to provide a depth field *i.e.* designed to capture 3D information rather than to establish location from this information.

Electro-Magnetic

These have a range of only about 1m, though more powerful systems can now cover a small room, and larger rooms if more transmitters are used. Systems are available from Polhemus [39], and the "Flock of Birds" from Ascension [40]. The technology is vulnerable to distortion due to the presence of large metal objects in the environment.

Inertial

"Inertial Proprioceptive Devices" are intelligent devices that know where they are. Work in this area is being done at MIT [41], where for example a conductor's baton that knows its position is proposed. This is one interpretation of the phrase "omnipresent computing" where intelligence is distributed into everyday objects. Two other interpretations are "wearables" where computing power is associated with an individual; and "net omnipresence" where the Internet escapes the confines of desktop computer nodes and finds its way into the fabric of everyday life. All possibilities are logical consequences of miniaturisation, though it remains to be seen by what combination of these mechanisms (if any) computing functionality will be distributed as the 21^{st} century progresses.

Solid state inertial measurement units (including accelerometers and gyroscopes) are produced by many manufacturers at relatively small cost. It is difficult to determine their suitability for position determination, as the error is not fixed but increases alarmingly as the square of time. This is due to the fact that a second order effect is being measured, and the pair of necessary integrations to compute position introduces the error. In just 60 seconds, a one-dimensional Inertial Measurement Unit (IMU) using an accelerometer with an output noise level of just 0.004 g yields a positional uncertainty of about 70 metres. A superb survey of inertial navigation techniques with all supporting mathematics is to be found in [42].

Hybrid Technologies

A hybrid sensor uses attributes of one detection system to counteract the deficiencies of another to create a composite virtual sensor. Let us consider the feasibility of a particular hybrid system for the sake of illustration. Can we build a cheap hybrid GPS and inertial system, where the inertial system provides fast relative position updates on an underlying more slowly updated GPS position fix? In order to answer this question, we need to examine how qualitative characteristics of real devices relate to practical situations.

Consider two IMUs fitted into the heels of a pair of shoes. Recorded accelerations for feet whilst walking fall within the range 0.19 g to 6.57 g, with nearly all the acceleration activity located near the mean of 1.59 g. Assume an IMU of < 0.004 g noise and a range of $\pm$ 10 g (such devices do exist). The measurement range is sufficient for the application, but what about the accuracy? Say we in-betweened a 1Hz GPS unit. Then the cumulated inertial positional error would be 2 cm between GPS measurements — which is probably acceptable for our AR application. As we speed up the cycle time of a position sensing system, the inertial performance improves dramatically. For example, the accelerometer with 0.004 g noise gives a positional uncertainty of about 0.2 mm per cycle in a system as slow as 10 Hz; the uncertainty falls to about 80 micrometers per cycle in a 50 Hz system. This is encouraging, and indeed a number of other references independently suggest that hybrid systems are the way of the future including [42] which suggests GPS and inertial and [3] which suggests optical and inertial.

One could envisage a hybrid orientation sensor which uses an IMU for accurate angular updates over short periods of time, but which relies on a longer cycle vision system for absolute corrections. The characteristics are complementary: IMUs tend to drift over time and computer vision processing

may take a significant time. In essence, one is continually re-calibrating the IMU. The lag in the vision processing must be taken into account, so the system calibrates itself retrospectively at the instance the image was captured.

One could also envisage a position/orientation sensor with three components: inertial (short-term relative angular and positional information); GPS (longer-term absolute positional information); and vision (longer-term absolute angular information). This is a simple-minded division of labour scenario, but there may be more general techniques of combining the different information sources in an optimum way.

The architectural AR system described in [6] suffers precisely from this registration problem that was anticipated. The user has to switch off temporarily the input from the digital compass and the differential GPS unit to freeze the on-screen display. The user then walks to the position where the system thinks it is, by aligning the virtual and real worlds, and finally switches on the sensors again. The differential GPS system was found to be accurate to 1 or 2 metres, but this rose to 3 or 5 metres next to buildings.

An electronic compass and a strapdown[2] IMU are used in [43] for outdoors AR, with differential GPS used for reference rather than in a hybrid fashion. However, this paper lacks any account of tracking accuracy, which is the hallmark of much work which discusses the issues but then "faults" at the hurdle of tackling the difficult registration problem head on. It is reported that the electronic compass can drift 5° over a few hours, and that this is one of the major problems with the system.

Hybrid Technologies with Application Knowledge

Knowledge of the application domain can compensate for sensor short-comings, and indeed an idea for IMU equipped shoes was generated as part of this project. If used as standalone raw accelerometers, IMUs in the heels of a pair of shoes will report a positional error that increases alarmingly with time. The novel principle is to use the natural time division of walking to provide greater accuracy for absolute position determination. Provided each step can be detected by signal processing (or alternatively by a micro-switch in each shoe) then it can be assumed at that instant the sensor is momentarily at rest. This knowledge can be used to reduce the drift in positional accuracy over time inherent in a device that measures acceleration. Relative position sensors on the two feet can "triangulate" at the instant that the stationary foot changes, and the locus followed in absolute space step-by-step.

[2] a *strapdown* inertial navigation system is one fixed rigidly to the moving body whose position is being measured

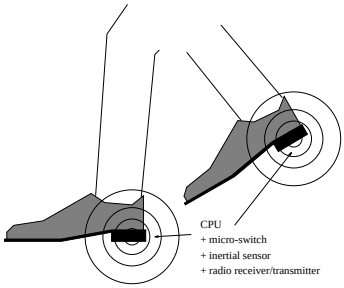

Figure 2.5: Sensor-equipped inter-communicating footwear

Dawdling will not cause the error to go unbounded as this can be limited to a human's step-size. An electronic compass could also contribute direction information acting in the manner of a hybrid sensor component. A schematic diagram of this proposed system is presented in Figure 2.5.

A BT project which transmitted radio signals from a tilt sensor connected to a mobile human subject allowed remote identification of the subject's activity through simple visual inspection of the trace. This is a more qualitative use of the same underlying technology. The idea of the IMU equipped shoes was subsequently located in a paper [44], though concept is presented rather than any results, implying either the system was not constructed (more likely) or that the results were bad and therefore withheld!

Despite good engineering practice, at the current state-of-the-art, it is not entirely wise to totally separate out sensor drivers and their use because, as explained, knowledge of the application domain can genuinely compensate for sensor short-comings. However, the subsystem of an instrumented pair of shoes, say, could be regarded as a higher level virtual position determining device. This illustrates the major challenge of AR in a nutshell: the expected software complexity, the unknown hardware composition, and current infeasibility of AR sensors prevents the production of interfaces and systems based on these.

Chapter 3

GPS Theory

3.1 Introduction

This chapter describes the theory behind the Global Positioning System (GPS) which is one of the technologies most suited to outdoor AR use. GPS is a complex, multi-component system and it is only by understanding the many disparate sources of error and informedly selecting appropriate algorithms and infrastructures that a GPS-based positioning technology with suitable characteristics for AR can be developed. In view of this complexity, concerns have been separated and the practical GPS work (*e.g.* selecting hardware, writing software and evaluating performance) is reported in the next chapter. There is no attempt to cover all of GPS theory here (and certainly there are excellent textbooks on the topic [45] and lecture notes [46]), however, sufficient theory is presented from the ground up to underpin the following practical work.

3.2 GPS Overview

The Global Positioning System (GPS) is based on a constellation of American Department of Defense satellites orbiting the Earth. A GPS receiver measures the time of flight of radio signals from the satellites within range. The time is measured by looking at the data content of the message which follows a known repeating pattern of pseudo-random numbers. The unique sequences from each satellite are synchronised by on-board atomic clocks to what is called the "GPS second". Given that the time of flight is but a small fraction of the repeat period, no range ambiguity is present. A receiver position fix, which is most consistent with these measured ranges, may be calculated by a least squares

technique. The position calculation is, in principle, little more than a simple triangulation problem. Accuracy depends on a number of factors: the most obvious is the number and configuration of the satellites in view. For example, if all the satellites are near the horizon, then the measured altitude will be more prone to error than otherwise.

The satellites broadcast on two frequencies called L1 (1575.42 MHz — largely for civilian use) and L2 (1227.60 MHz — largely for military use as the signals are encrypted). The timing information for the civilian signal has historically been deliberately degraded in accuracy resulting in a slowly varying random error in positional fix. When this happens Selective Availability (or SA) is described as being "on". In fact, SA was switched off permanently and somewhat unexpectedly on May 1^{st} 2000 whilst our GPS system was under development. This event naturally changed the "playing field" dramatically, and its consequences for our system and other position-based applications are discussed in the next chapter. A guideline nominal accuracy for GPS with SA on is 100m and 10m with SA off.

The biggest source of error, aside from SA when present, is due to the time and location varying propagation delay of the radio signals though the atmosphere. The product of the flight time of the radio waves and c (the speed of light in a vacuum) is called the pseudorange, and this will be inconveniently different from the actual range leading to error in the GPS position fix. A variety of enhancement techniques are available to, at least partially, circumvent the problem:

1. Modelling the Atmosphere
2. Dual Frequency Reception
3. Differential Operation

These are described below in the following subsections.

3.2.1 Modelling the Atmosphere

The satellites broadcast parameters for a (time varying) atmospheric model which allows some corrections to be calculated. These corrections are applied to the measured pseudoranges.

3.2.2 Dual Frequency Reception

If both L1 and L2 signals are received, the effects of atmospheric delay can be compensated for because the relative delay is described by a factor $\gamma = \frac{{L1}^2}{{L2}^2}$. Generally, only military class hardware is dual frequency.

3.2.3 Differential Operation

By using a second receiver at a known fixed location, a technique known as “differential GPS” can be employed. Provided the base/reference station and the mobile/remote station are sufficiently close together (say within 50km) it can be assumed that the signal delays at both locations from the same satellite are identical. As the location of the base station is known this delay can be calculated and used to correct the pseudoranges for the mobile station. The error for differential GPS is generally quoted as being some proportion of the baseline distance between the fixed and mobile stations. Perhaps surprisingly differential GPS calculations are simpler than for a single receiver, because many errors are exactly cancelled out. For example, it is neither necessary to apply atmospheric model corrections nor to use the different propagation properties of the L1 and L2 frequencies. Indeed, differential operation will cancel out the effect of SA — which, of course, is why the technique was devised.

Basic differential GPS is called “differential code GPS” because the message content is used to measure journey time. An enhancement is called “differential phase GPS” where the phase of the carrier signal is also used to obtain a more accurate pseudorange measurement. The phase measurement is continuously monitored to produce a quantity called the “integrated carrier phase” which incorporates all detected phase wrap-arounds. Even the integrated carrier phase can only provide a relative distance measurement, and phase wrap-arounds that are not detected (known as “cycle slips”) lead to errors over time in the value of this integrated quantity. Intelligent filtering techniques are used to combine both code and phase measurements to obtain the best aspects of either, yielding a more accurate yet absolute measure. The error for differential phase GPS is generally quoted as being as low as 1 ppm of the baseline distance between the fixed and mobile stations. The downsides to differential GPS are threefold:

1. a second GPS receiver is required;
2. the location of this receiver must be known accurately;
3. the information from this receiver must be propagated to the mobile station in “real-time”: an upper limit on the propagation time of 15 seconds has been quoted. With SA switched off there is less urgency in propagating the differential correction.

Due to these common requirements and the high set-up overhead involved, there are a number of differential GPS correction service providers who broadcast pseudorange adjustments by satellite, FM radio and by maritime beacon in a format which adheres to the RTCM 104 standard [47].

3.3 Forming Differences to Eliminate Errors

The primary input for a GPS position calculation is a set of pseudorange measurements. If these pseudorange measurements accurately reflected the true ranges between the satellites and the GPS receiver, then the position fix achieved would be highly accurate. The only sources of error would be the intrinsic noise in the measurement and the uncertainty in the satellite positions. Even with a cheap GPS receiver, the overall positional error would be just about low enough for AR. However, a pseudorange measurement is subject in practice to a variety of sources of significant systematic error, and a failure to acknowledge or model these errors in calculations would totally throw out a position fix. In fact, virtually all enhanced techniques for GPS processing are concerned with refining the accuracy of the raw pseudorange measurements. The error is modelled both between the actual and measured range, and the actual and measured "carrier range". A carrier range is just the carrier phase expressed in terms of distance, and is given by:

$$C = R + X_S - I + T + E_S + E_R + M_C + S_C + L \times N$$

where C is the carrier range (measured); R is the actual range between satellite and receiver; X_S is the error in calculated satellite position (largely due to SA if present); I is the ionospheric delay; T is the tropospheric delay; E_S is the satellite clock error (largely due to SA if present); E_R is the receiver clock error; M_C is the multipath on carrier range measurement; S_C is the noise on carrier range measurement; L is the GPS carrier wavelength ($\approx$ 19 cm); and N is the carrier phase integer ambiguity. The pseudorange is given by a somewhat similar expression:

$$P = R + X_S + I + T + E_S + E_R + M_P + S_P$$

where P is the pseudorange (measured); M_P is the multipath on pseudorange measurement (due to reflections); and S_P is the noise on pseudorange measurement. Forming various differences between measurements of P and C allows some errors to be cancelled. Within GPS terminology, single, double and triple differencing have the precise meanings given below, even though many such differences could be computed.

1. **Difference Between Pseudorange and Carrier Range.** For the same satellite at a single receiver:

$$\begin{aligned} P - C &= 2 \times I + M_P - M_C + S_P - S_C - L \times N \\ &\approx 2 \times I + M_P + S_P - L \times N \qquad \text{as } M_C \ll M_P \text{ and } S_C \ll S_P \end{aligned}$$

2. **Single Difference.** The ranges (be it carrier range or pseudorange) from two receivers R_1 and R_2 to the same satellite are subtracted (Figure 3.1) to give:

$$\begin{aligned} dP &= P_{R_1} - P_{R_2} = dR + dI + dT + dE_R + dM_P + dS_P \\ dC &= C_{R_1} - C_{R_2} = dR - dI + dT + dE_R + dM_C + dS_C + L \times dN \end{aligned}$$

   The important property of this single difference is that the errors due to the satellite clock and satellite position cancel. SA if present is therefore removed. In order to cancel the time varying errors adequately, the observations of both receivers have to be made at nearly the same moment. This is a few microseconds for the pseudorange range, but a few nanoseconds for the carrier range. For short baselines (say less than 20 km), dT and dI may be ignored.

3. **Double Difference.** The single differences for two different satellites S_1 and S_2 using the same two receivers are subtracted (Figure 3.2). Usually S_1 is chosen to be the highest of all the satellites detected because this will give the most reliable pseudorange measurements.

$$\begin{aligned} ddP &= (P_{S_1R_1} - P_{S_1R_2}) - (P_{S_2R_1} - P_{S_2R_2}) \\ &= ddR + ddI + ddT + ddM_P + ddS_P \\ ddC &= (C_{S_1R_1} - C_{S_1R_2}) - (C_{S_2R_1} - C_{S_2R_2}) \\ &= ddR + ddI + ddT + ddM_C + ddS_C + L \times ddN \end{aligned}$$

   The double differences remove the receivers' clock errors. For short baselines ddI and ddT may again be neglected. Thus:

$$\begin{aligned} ddP &\approx ddR + ddM_P + ddS_P \\ ddC &\approx ddR + ddM_C + ddS_C + L \times ddN \end{aligned}$$

The expression for ddC is used for high accuracy differential carrier phase GPS. For example, with two identical receivers at the same location, ddR, ddM_C and ddM_P are all zero, so:

$$ddC \approx L \times ddN$$

Aside from noise, ddC is equal to an integer number of signal wavelengths. With sufficiently good equipment (*i.e.* low noise) it should be possible to estimate the integer value of ddN and thereby actually determine the noise in the system, leading to far greater accuracy. Naturally, when the baseline is non-zero the processing becomes much more complex.

4. **Triple Difference.** Two double differences at different times (or "epochs") are subtracted (Figure 3.3) to give:

$$\begin{aligned} dddP &= [(P_{S_1R_1t_1} - P_{S_1R_2t_1}) - (P_{S_2R_1t_1} - P_{S_2R_2t_1})] - \\ &\quad [(P_{S_1R_1t_2} - P_{S_1R_2t_2}) - (P_{S_2R_1t_2} - P_{S_2R_2t_2})] \\ dddC &= [(C_{S_1R_1t_1} - C_{S_1R_2t_1}) - (C_{S_2R_1t_1} - C_{S_2R_2t_1})] - \\ &\quad [(C_{S_1R_1t_2} - C_{S_1R_2t_2}) - (C_{S_2R_1t_2} - C_{S_2R_2t_2})] \end{aligned}$$

Assuming no cycle slips or loss of lock, triple differencing removes the integer ambiguity present in carrier range and the value of $dddP$ and $dddC$ (modulo noise) should be zero. Thus triple difference can be used to detect cycle slips and loss of lock.

3.4 Position Calculation Overview

The outline algorithm for all GPS position calculations is as follows:

1. calculate satellite positions at time of message transmission
2. correct positions for rotation of Earth during transmission
3. calculate azimuth and elevation of satellite
4. correct pseudoranges, for such factors as:

   (a) ionospheric delay (depends on satellite azimuth and elevation)

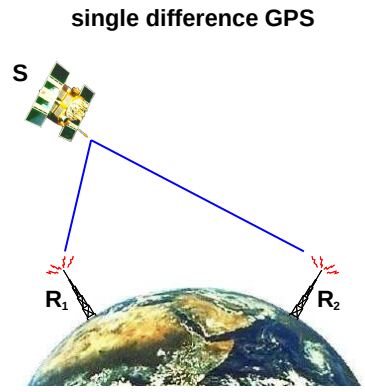

Figure 3.1: Single difference GPS

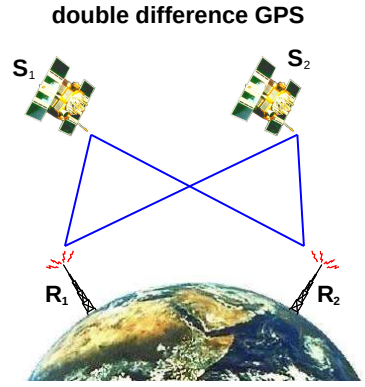

Figure 3.2: Double difference GPS

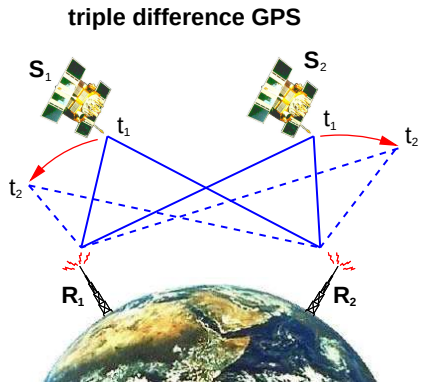

Figure 3.3: Triple difference GPS

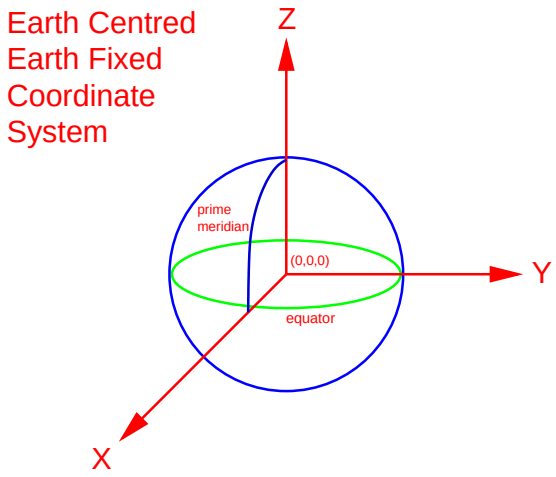

Figure 3.4: ECEF coordinate system

(b) tropospheric delay (depends on satellite azimuth and elevation)

(c) clock error

5. calculate receiver position from corrected pseudoranges

In fact this algorithm remains the same, irrespective of whether single or dual receivers are in use and irrespective of whether carrier phase is also being used. However, the details of the pseudorange correction will vary.

3.5 Position Calculation Particulars

This section gives a full formal mathematical treatment of the GPS position calculation, as instanced in Sam Storm van Leeuwen's public domain Pascal code which can be downloaded from the Web [48]. This is provided so the subsequent modifications to the method described below and in Chapter 4 have a formal context in which to be presented. The component calculations in the GPS position determination are provided below, in a standard tabular format to make clear which values are constants, which are input parameters and which are derived.

The coordinate system used in GPS in called the ECEF (Earth Centred, Earth Fixed) system (Figure 3.4). The origin is fixed at the centre of the Earth; the Z axis sticks out of the North Pole;

the X axis sticks out where the Greenwich Meridian crosses the Equator; and the Y axis is defined implicitly by $\hat{Z} \times \hat{X}$ as the system is right-handed.

3.5.1 Calculate Satellite Positions

The first stage in a GPS position calculation is to work out the whereabouts of the satellites. These are the foundation points in relation to which the location of the GPS receiver is determined. Note that satellite position in theory is purely a deterministic function of t, as satellites, being in a vacuum, follow simple Newtonian laws. In practice, the orbits are more complex as the geoid (gravitational equipotential) is complex, and this leads to the other input parameters, which describe the orbit of each satellite. These are placed in a separate data table because they follow a different data path, being broadcast by each satellite as "ephemeris data" rather than measured *per se*.

Constants		
symbol	description	value
μ	Earth's universal gravitational parameter	$3.986005 \times 10^{14} \mathrm{m}^3\mathrm{s}^{-2}$
$\dot{\Omega}_e$	Earth's rotation rate	$7.2921151467 \times 10^{-5}\,\mathrm{s}^{-1}$ (in radians)

Output Result		
symbol	description	unit
(x_k, y_k, z_k)	Earth-fixed coordinates	$(\mathrm{m}, \mathrm{m}, \mathrm{m})$

Input Parameters		
symbol	description	unit
t	GPS system time at transmission	s

Input Parameters (Ephemeris)		
symbol	description	unit
$A^{\frac{1}{2}}$	semi-major axis of orbit	$\mathrm{m}^{\frac{1}{2}}$
t_{oe}	ephemeris epoch time	s
M_0	mean anomaly at reference time	dimensionless
e	eccentricity	dimensionless
ω	argument of perigee	dimensionless
C_{uc}	amplitude of cosine harmonic correction term to the argument of latitude	dimensionless
C_{us}	amplitude of sine harmonic correction term to the argument of latitude	dimensionless
C_{rc}	amplitude of cosine harmonic correction term to the orbit radius	m
C_{rs}	amplitude of sine harmonic correction term to the orbit radius	m
C_{ic}	amplitude of cosine harmonic correction term to the angle of inclination	m
C_{is}	amplitude of sine harmonic correction term to the angle of inclination	m
Ω_0	longitude of ascending node of orbit plane at weekly epoch	dimensionless
$\dot{\Omega}$	rate of right ascension	s^{-1}
Δn	mean motion difference from computed value	s^{-1}
i_0	inclination angle a reference time	dimensionless
$\dot{i}$	rate of inclination angle	s^{-1}

Variables		
symbol	description	unit
t_k	time from ephemeris reference epoch	s
n_0	computed mean motion	s^{-1} (radians)
n	corrected mean motion	s^{-1} (radians)
M_k	mean anomaly	dimensionless
E_k	eccentric anomaly	dimensionless
v_k	true anomaly	dimensionless
Φ_k	argument of latitude	dimensionless
δu_k	argument of latitude correction	dimensionless
δu_r	radius correction	m
δu_i	inclination correction	dimensionless
u_k	corrected argument of latitude	dimensionless
u_r	corrected radius	m
u_i	corrected inclination	dimensionless
x'_k, y'_k	position in orbital plane	m
Ω_k	corrected longitude of ascending node	dimensionless

$$
\begin{aligned}
A &= (\sqrt{A})^2 \\
n_0 &= \sqrt{\frac{\mu}{A^3}} \\
t_k &= t - t^*_{oe} \\
n &= n_0 + \Delta n \\
M_k &= M_0 + n t_k \\
M_k &= E_k - e \sin E_k \quad \text{(solve for } E_k \text{ by using simple iteration)}
\end{aligned}
$$

$$
\begin{aligned}
v_k &= \tan^{-1}(\cos v_k, \sin v_k) \\
&= \tan^{-1}\left(\frac{\cos E_k - e}{1 - e\cos E_k}, \frac{\sqrt{1-e^2}\sin E_k}{1 - e\cos E_k}\right) \\
\Phi_k &= v_k + \omega \\
\delta u_k &= c_{us}\sin 2\Phi_k + c_{uc}\cos 2\Phi_k \\
\delta r_k &= c_{rs}\sin 2\Phi_k + c_{rc}\cos 2\Phi_k \\
\delta i_k &= c_{is}\sin 2\Phi_k + c_{ic}\cos 2\Phi_k \\
u_k &= \Phi_k + \delta u_k \\
r_k &= A(1 - e\cos E_k) + \delta r_k \\
i_k &= i_0 + \delta i_k + \dot{i}t_k \\
x'_k &= r_k \cos u_k \\
y'_k &= r_k \sin u_k \\
\Omega_k &= \Omega_0 + (\dot{\Omega} - \dot{\Omega}_e)t_k - \dot{\Omega}_e t_{oe} \\
x_k &= x'_k \cos\Omega_k - y'_k \cos i_k \sin\Omega_k \\
y_k &= x'_k \sin\Omega_k + y'_k \cos i_k \cos\Omega_k \\
z_k &= y'_k \sin i_k
\end{aligned}
$$

3.5.2 Converting ECEF Coordinates to Latitude and Longitude

The algorithms to interconvert between these two different coordinates systems are cosmetic in the sense that the entire GPS position fix calculation could be performed purely in the ECEF coordinate system. However, humans are more comfortable with latitude and longitude, and therefore it is a more convenient form in which to present the final calculated position. Confusingly, there are a great number of latitude and longitude conventions. We shall be concerned with only two: WGS 84 — a global standard which is the main one for the GPS community; and Airy 1936 in use in Britain and Ireland by the Ordnance Survey.

Constants			
symbol	description	WGS-84 Values	Airy 1936 Values
a	semi-major axis	6378137.0000 m	6377563.3960 m
f^{-1}	inverse flattening	298.257223563	299.324964600
$(\delta X, \delta Y, \delta Z)$	datum offset	(0 m,0 m,0 m)	(375 m,−111 m,431 m)

Output Results		
symbol	description	unit
$Latitude$	latitude	radians
$Longitude$	longitude	radians
$Altitude$	altitude	m

Input Parameter		
symbol	description	unit
(X', Y', Z')	ECEF coordinates	$(\mathrm{m}, \mathrm{m}, \mathrm{m})$

Variables		
symbol	description	unit
(X, Y, Z)	datum corrected ECEF coordinates	m
p	distance from Earth's axis to point	m
e_1	first numerical eccentricity	dimensionless
e_2	second numerical eccentricity	dimensionless
N	radius of curvature in prime vertical	m
θ	unknown	dimensionless

$$
\begin{aligned}
X &= X' - \delta X \\
Y &= Y' - \delta Y \\
Z &= Z' - \delta Z \\
b &= a \times \left(\frac{f^{-1} - 1}{f^{-1}} \right) \quad \text{flattening } f \text{ defined as } \tfrac{a-b}{a} \\
e_1 &= \frac{a^2 - b^2}{a^2} \\
e_2 &= \frac{a^2 - b^2}{b^2} \\
p &= \sqrt{X^2 + Y^2} \\
\theta &= \tan^{-1}\left(\frac{Za}{pb} \right) \\
Latitude &= \tan^{-1}\left(\frac{Z + e_2 b \sin^3\theta}{p - e_1 a \cos^3\theta} \right) \\
Longitude &= \frac{\pi}{2} \quad \text{if } x = 0 \\
Longitude &= \tan^{-1}\frac{Y}{X} \quad \text{otherwise} \\
N &= \frac{a}{\sqrt{1 - e_1 * \sin^2(Latitude)}} \\
Altitude &= \frac{p}{\cos(Latitude)} - N
\end{aligned}
$$

3.5.3 Converting Latitude and Longitude to ECEF Coordinates

This conversion allows the user-supplied initial "guess" at the GPS receiver's position (necessary for the overall algorithm's operation) to be specified conveniently as latitude and longitude. Note that there is no necessity for this guess to be accurate.

Input Parameters		
symbol	description	unit
$Latitude$	latitude	radians
$Longitude$	longitude	radians
$Altitude$	altitude	m

Output Result		
symbol	description	unit
(X', Y', Z')	ECEF coordinates	$(\mathrm{m}, \mathrm{m}, \mathrm{m})$

Variable		
symbol	description	unit
N	distance from Earth's centre to surface	m

$$
\begin{aligned}
N &= \frac{a}{1 - e_1 \sin^2(Latitude)} \\
X' &= (N + Altitude)\cos(Latitude)\cos(Longitude) + \delta X \\
Y' &= (N + Altitude)\cos(Latitude)\sin(Longitude) + \delta Y \\
Z' &= (N(1 - e_1) + Altitude)\sin(Latitude) + \delta Z
\end{aligned}
$$

3.5.4 Calculating Azimuth and Elevation

To correct the measured pseudoranges for the effects of the atmosphere, it is necessary to know the angular bearing (both azimuth and elevation) of each satellite with respect to the GPS receiver.

Input Parameter		
symbol	description	unit
(X_r, Y_r, Z_r)	receiver position	(m,m,m)
(X_s, Y_s, Z_s)	satellite position	(m,m,m)

Output Results		
symbol	description	unit
$Elevation$	elevation of satellite from receiver	dimensionless (radians)
$Azimuth$	azimuth of satellite from receiver	dimensionless (radians)

Variables		
symbol	description	unit
p	perpendicular from Earth's axis to receiver	m
R	distance from centre of Earth to receiver	dimensionless
$\boldsymbol{east}$	unit vector at receiver pointing east	$(\mathrm{m}, \mathrm{m}, \mathrm{m})$
$\boldsymbol{north}$	unit vector at receiver pointing north	$(\mathrm{m}, \mathrm{m}, \mathrm{m})$
$\boldsymbol{up}$	unit vector at receiver pointing up	$(\mathrm{m}, \mathrm{m}, \mathrm{m})$
$\boldsymbol{d}$	receiver to satellite unit vector	$(\mathrm{m}, \mathrm{m}, \mathrm{m})$
D	receiver to satellite distance	m
$\boldsymbol{d'}$	$\boldsymbol{d}$ in receiver's coordinate system	$(\mathrm{m}, \mathrm{m}, \mathrm{m})$

$$
\begin{aligned}
p &= \sqrt{X_r^2 + Y_r^2} \\
R &= \sqrt{X_r^2 + Y_r^2 + Z_r^2} \\
\boldsymbol{east} &= \left(-\frac{Y_r}{p}, \frac{X_r}{p}, 0\right) \\
\boldsymbol{up} &= \left(\frac{X_r}{R}, \frac{Y_r}{R}, \frac{Z_r}{R}\right) \\
\boldsymbol{north} &= \boldsymbol{up} \times \boldsymbol{east} \\
&= \left(-\frac{X_r \times Z_r}{p \times R}, -\frac{Y_r \times Z_r}{p \times R}, \frac{p}{R}\right)
\end{aligned}
$$

The unit direction vector from the receiver to the satellite is:

$$
\boldsymbol{d} = \left(\frac{X_s - X_r}{D}, \frac{Y_s - Y_r}{D}, \frac{Z_s - Z_r}{D}\right) \quad \text{where} \quad D = \sqrt{(X_s - X_r)^2 + (Y_s - Y_r)^2 + (Z_s - Z_r)^2}
$$

In the coordinate system defined by $(\boldsymbol{east}, \boldsymbol{north}, \boldsymbol{up})$ the receiver $\rightarrow$ satellite unit vector is:

$$
\boldsymbol{d'} = \begin{bmatrix} -\frac{Y_r}{p} & \frac{X_r}{p} & 0 \\ -\frac{X_r \times Z_r}{p \times R} & -\frac{Y_r \times Z_r}{p \times R} & \frac{p}{R} \\ \frac{X_r}{R} & \frac{Y_r}{R} & \frac{Z_r}{R} \end{bmatrix} \begin{bmatrix} \frac{X_s - X_r}{D} \\ \frac{Y_s - Y_r}{D} \\ \frac{Z_s - Z_r}{D} \end{bmatrix}
$$

$$
\begin{aligned}
Elevation &= \tan^{-1}(\sqrt{{d'}_x^2 + {d'}_y^2}, d'_z) \\
Azimuth &= \tan^{-1}({d'}_x, {d'}_y) + \pi
\end{aligned}
$$

3.5.5 Calculating Ionospheric Delay

The temporal delay in signal propagation introduced by the ionosphere is calculated. This will be used later to correct the measured pseudorange.

Input Parameters		
symbol	description	unit
T_{tr}	time of transmission	s
$Elevation$	elevation of satellite from receiver	dimensionless (radians)
$Azimuth$	azimuth of satellite from receiver	dimensionless (radians)
b_0	ionospheric parameter	s
b_1	ionospheric parameter	$\frac{\text{s}}{(\text{semi-circles})}$
b_2	ionospheric parameter	$\frac{\text{s}}{(\text{semi-circles})^2}$
b_3	ionospheric parameter	$\frac{\text{s}}{(\text{semi-circles})^3}$
a_0	ionospheric parameter	s
a_1	ionospheric parameter	$\frac{\text{s}}{(\text{semi-circles})}$
a_2	ionospheric parameter	$\frac{\text{s}}{(\text{semi-circles})^2}$
a_3	ionospheric parameter	$\frac{\text{s}}{(\text{semi-circles})^3}$

Output Result		
symbol	description	unit
dT_{iono}	ionospheric delay correction	s

Variables		
symbol	description	unit
$Latitude_i$	sub-ionospheric latitude	semi-circles
$Latitude'_i$	corrected sub-ionospheric latitude	semi-circles
$Latitude_m$	geometric latitude of the ionospheric intersection point	semi-circles
$Longitude_i$	sub-ionospheric longitude	semi-circles
F	slant factor	dimensionless
ϕ	Earth-centred angle	semi-circles
T	local time at sub-ionospheric point	s
T'	corrected local time at sub-ionospheric point	s
per	temporary variable	s
per'	corrected per	s
amp	temporary variable	s
amp'	corrected amp	s
x	"x"	radians

$$\begin{aligned} \phi &= \frac{0.00137}{(Elevation + 0.11) - 0.022} \\ Latitude_i &= Latitude + \phi \cos(\pi Azimuth) \end{aligned}$$

$$
\begin{aligned}
Latitude'_i &= 0.416 \quad \text{if } Latitude_i > +0.416 \\
&= -0.416 \quad \text{if } Latitude_i < -0.416 \\
Longitude_i &= Longitude + \phi \frac{\sin(\pi Azimuth)}{\cos(\pi Latitude'_i)} \\
Latitude_m &= Latitude_i + 0.064 * \cos(\pi(Longitude_i - 1.617)) \\
T &= 43200 + Longitude_i * T_{tr} \\
T' &= T - 86400 \quad \text{if } T > 86400 \\
&= T + 86400 \quad \text{if } T < 0 \\
F &= 1 + 16(0.53 - Elevation)^3 \\
per &= b_0 + b_1 Latitude_m + b_2 Latitude_m^2 + b_3 Latitude_m^3 \\
per' &= 72000 \quad \text{if } per < 72000 \\
&= per \quad \text{otherwise} \\
x &= 2\pi \frac{T' - 50400}{per'} \\
amp &= a_0 + a_1 Latitude_m + a_2 Latitude_m^2 + a_3 Latitude_m^3 \\
amp' &= 0 \quad \text{if } amp < 0 \\
&= amp \quad \text{otherwise} \\
dT_{iono} &= F\left(5.0 \times 10^{-9}\right) \quad \text{if } |x| \geq 1.57 \\
&= F\left(5.0 \times 10^{-9} + amp\left(1 - \frac{x^2}{2} + \frac{x^4}{24}\right)\right) \quad \text{otherwise}
\end{aligned}
$$

3.5.6 Calculating Tropospheric Delay

The spatial delay in signal propagation introduced by the troposphere is calculated. This will be used later to correct the measured pseudorange.

Input Parameter		
symbol	description	unit
$Elevation$	elevation of satellite from receiver	dimensionless (radians)

Output Result		
symbol	description	unit
dR_{trop}	tropospheric delay	m

$$
dR_{trop} = \frac{2.312}{\sin(\sqrt{Elevation^2 + 1.904 \times 10^{-3}})} + \frac{0.084}{\sin(\sqrt{Elevation^2 + 0.6854 \times 10^{-3}})}
$$

3.5.7 Correcting Satellite Clock

The error in each satellite's clock is computed. This is later used to correct the pseudorange measurements. The satellites broadcast parameters which model their own clock drifts, and these are used in the calculations.

Constants		
symbol	description	value
F	constant	$-4.442807633 \times 10^{-10}\text{sm}^{-0.5}$

Input Parameters		
symbol	description	unit
T_{tr}	time of transmission	s
T_{oc}	SV clock reference time	s
T_{gd}	group delay	s
a_{f0}	zeroth order polynomial coefficient	s
a_{f1}	first order polynomial coefficient	dimensionless
a_{f2}	second order polynomial coefficient	s^{-1}
$\sqrt{A}$	root of semi-major axis of SV	$\text{m}^{-0.5}$
e	SV orbit eccentricity	dimensionless
E	SV orbit eccentric anomaly	radians

Output Parameter		
symbol	description	unit
dT_{clock}	SV clock correction	s

Variables		
symbol	description	unit
T	time since reference time	s

$$
\begin{aligned}
T &= (T_{Tr} - T_{oc}) - 604800 \quad \text{if } (T_{Tr} - T_{oc}) > 302400 \\
&= (T_{Tr} - T_{oc}) + 604800 \quad \text{if } (T_{Tr} - T_{oc}) < -302400 \\
&= (T_{Tr} - T_{oc)} \quad \text{otherwise} \\
dT_{clock} &= a_{f0} + a_{f1}T + a_{f2}T^2 + F * e + \sqrt{A} * \sin(E) - T_{gd}
\end{aligned}
$$

3.5.8 Correcting Pseudorange

The measured pseudorange is corrected for the delaying effects of the ionosphere and troposphere and corrected for the error in the satellite's clock.

Constant		
symbol	description	value
c	speed of light	$299792458.0\mathrm{ms}^{-1}$

Input Parameters		
symbol	description	unit
PR	measured pseudorange	m
dT_{clock}	SV clock correction	s
dT_{iono}	ionospheric delay correction	s
dR_{trop}	tropospheric delay	m

Output Parameter		
symbol	description	unit
PR_{corr}	corrected pseudorange	m

$$PR_{corr} \quad = \quad PR + dT_{clock} \times c - dT_{iono} \times c - dR_{trop}$$

3.5.9 Solving for Receiver Position

Note that this GPS calculation is more complex than those presented so far, as a system of equations have to be solved simultaneously. Both the derivation and solution of this system of equations are presented.

Input Parameters		
symbol	description	unit
P_{rs}	pseudoranges (after potential correction)	m
$\boldsymbol{X}_r$	approximate receiver position	$(\mathrm{m}, \mathrm{m}, \mathrm{m})$
$\boldsymbol{X}_s$	satellite positions	$(\mathrm{m}, \mathrm{m}, \mathrm{m})$

Output Results		
symbol	description	unit
$\boldsymbol{d_X}$	correction to approximate receiver position	$(\mathrm{m}, \mathrm{m}, \mathrm{m})$
C_r	error in receiver clock	m

Variables		
symbol	description	unit
E_s	remaining errors in pseudorange	m
R_{rs}	range from satellite to approximate receiver position	m

For each satellite s:

$$P_{rs} + C_r + E_s \quad = \quad \sqrt{\boldsymbol{X}'^{\,2}_{rs}}$$

Note that $\boldsymbol{X}'_{rs} = \boldsymbol{X}_r - \boldsymbol{X}_s$ is the actual satellite to receiver vector. Now let $\boldsymbol{X}'_r = \boldsymbol{X}_r + \boldsymbol{d_X}$ where $\boldsymbol{X}_r$ is the approximately known receiver position and $\boldsymbol{d_X}$ the correction.

$$P_{rs} + C_r + E_s \quad = \quad \sqrt{((\boldsymbol{X_r} + \boldsymbol{d_X}) - \boldsymbol{X_s})^2} \tag{3.1}$$

$$\begin{aligned}
P_{rs} + C_r + E_s &= \sqrt{(\boldsymbol{X_r} - \boldsymbol{X_s})^2 + 2(\boldsymbol{X_r} - \boldsymbol{X_s}).\boldsymbol{d_X} + \boldsymbol{d_X}^2} \\
&= \sqrt{R_{rs}^2 + 2(\boldsymbol{X_r} - \boldsymbol{X_s}).\boldsymbol{d_X} + \boldsymbol{d_X}^2} \quad \text{where } R_{rs} \text{ is the approximated range} \\
&= R_{rs}\sqrt{1 + \frac{2(\boldsymbol{X_r} - \boldsymbol{X_s}.)\boldsymbol{d_X}}{R_{rs}^2} + \frac{\boldsymbol{d_X}^2}{R_{rs}^2}} \\
&= R_{rs}\sqrt{1 + \frac{2(\boldsymbol{X_r} - \boldsymbol{X_s}).\boldsymbol{d_X}}{R_{rs}^2} + \left(\frac{(\boldsymbol{X_r} - \boldsymbol{X_s}).\boldsymbol{d_X}}{R_{rs}^2}\right)^2 + \frac{\boldsymbol{d_X}^2}{R_{rs}^2} - \left(\frac{(\boldsymbol{X_r} - \boldsymbol{X_s}).\boldsymbol{d_X}}{R_{rs}^2}\right)^2} \\
&= R_{rs}\sqrt{1 + 2a + a^2 + e} \text{ where } a = \tfrac{(\boldsymbol{X_r} - \boldsymbol{X_s}).\boldsymbol{d_X}}{R_{rs}^2}\ ,\ e = \tfrac{\boldsymbol{d_X}^2}{R_{rs}^2} - \left(\tfrac{(\boldsymbol{X_r} - \boldsymbol{X_s}).\boldsymbol{d_X}}{R_{rs}^2}\right)^2 \\
&\approx R_{rs}\sqrt{(1+a)^2} \quad \text{as e contains higher order terms in } \boldsymbol{d_X} \\
&\approx R_{rs} + \frac{(\boldsymbol{X_r} - \boldsymbol{X_s}).\boldsymbol{d_X}}{R_{rs}}
\end{aligned}$$

Placing unknowns $\boldsymbol{d_X} = (d_X, d_Y, d_Z)$ and C_r on the LHS

$$\frac{(\boldsymbol{X_r} - \boldsymbol{X_s}).\boldsymbol{d_X}}{R_{rs}} - Cr \quad \approx \quad P_{rs} - R_{rs} + E_s$$

E_s is taken to be zero, R_{rs} is approximated from roughly known satellite and receiver positions. We thus set up an over-constrained linear system of equations:

$$\begin{bmatrix} \frac{X_r - X_{s_1}}{R_{rs_1}} & \frac{Y_r - Y_{s_1}}{R_{rs_1}} & \frac{Z_r - Z_{s_1}}{R_{rs_1}} & -1 \\ \frac{X_r - X_{s_2}}{R_{rs_2}} & \frac{Y_r - Y_{s_2}}{R_{rs_2}} & \frac{Z_r - Z_{s_2}}{R_{rs_2}} & -1 \\ \vdots & \vdots & \vdots & \vdots \\ \frac{X_r - X_{s_{32}}}{R_{rs_{32}}} & \frac{Y_r - Y_{s_{32}}}{R_{rs_{32}}} & \frac{Z_r - Z_{s_{32}}}{R_{rs_{32}}} & -1 \end{bmatrix} \begin{bmatrix} d_X \\ d_Y \\ d_Z \\ C_r \end{bmatrix} \approx \begin{bmatrix} P_{rs_1} - R_{rs_1} \\ P_{rs_2} - R_{rs_2} \\ \vdots \\ P_{rs_{32}} - R_{rs_{32}} \end{bmatrix}$$

Multiply both sides by $(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$ where $\boldsymbol{A}$ is the coefficient matrix. We obtain a solution:

$$\begin{bmatrix} d_X \\ d_Y \\ d_Z \\ C_r \end{bmatrix} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T \begin{bmatrix} P_{rs_1} - R_{rs_1} \\ P_{rs_2} - R_{rs_2} \\ \vdots \\ P_{rs_{32}} - R_{rs_{32}} \end{bmatrix}$$

Iteration of this solution is defined by:

$$\begin{aligned} (\boldsymbol{X_r})_{n+1} &= (\boldsymbol{X_r})_n + \boldsymbol{d_X} \quad \text{update the approximate position using the correction} \\ (\boldsymbol{X_r})_0 &= \boldsymbol{X}_{London} \quad \text{obtained from atlas} \end{aligned}$$

Iteration occurs until the position correction falls below the centimetre level:

$$|d_Z| + |d_Y| + |d_Z| < 0.01$$

3.6 Two-Pass Iteration for Single Receiver

In practice the positions of the satellites are not known absolutely as the time of transmission of each signal is not known directly. Instead, the GPS receiver tags the measured pseudoranges with the time of reception T_{rc} and the time of transmission T_{tr} has to be somehow inferred. Unfortunately, this is **not** simply a case of subtracting the pseudorange in seconds *i.e.* $T_{tr} \neq T_{rc} - \frac{P}{c}$ because the timer in the GPS receiver may be out of synchronisation with global GPS time. A standard GPS position fix calculation also solves for the error in the GPS unit's clock (based on a quartz crystal rather than the atomic clock of the satellite), because even a small clock error translates into a large corresponding range error C_r given the speed of light. To move towards a practical solution to determine satellite position, it is worthwhile describing the three categories of in-built clock on GPS units:

1. Type 1 receivers keep their internal clock continuously synced to the GPS time.

2. Type 2 receivers reset their clock once it has drifted away a few milliseconds from GPS time.

3. Type 3 receivers let their internal clock drift away from GPS time forever (usually they reset their internal clock after power-up once the first valid position fix has been calculated).

Two types of GPS receivers are used in the practical work: the Marconi AllStar unit is Type 1 and the Garmin G12XL unit is Type 3 (see the next chapter). Whatever the class of receiver, it is necessary to compute a clock correction by going through an initial position fix iteration, before performing a second and definitive position calculation based on proper satellite positions derived from a corrected value of of T_{rc}. The average transmission time is $\tau = 0.075$ s (based on the average distance between a satellite and an arbitrary point on Earth's surface) and this can be assumed for the first pass. Each of the two iterations is as follows and uses as components the particular calculations given earlier:

For each satellite:

- let $T_{tr} = T_{rc} - \tau$
- calculate satellite positions at estimated time of transmission T_{tr}
- correct satellite positions for rotation of Earth by $\tau\dot{\Omega}_e$ during transmission
- calculate azimuth and elevation of satellites
- calculate pseudorange corrections dT_{clock}, dT_{iono} and dR_{trop} based on these
- correct the pseudorange: $P' = P + dT_{clock} \times c - dT_{iono} \times c - dR_{trop}$

Globally:

- calculate receiver position and clock correction C_r

For each satellite:

- correct time of transmission $\tau' = \frac{P'+C_r}{c}$
- correct time of reception $T'_{rc} = T_{rc} + \frac{C_r}{c}$

3.7 Dual Receiver Operation

Consider both a fixed reference GPS base station at a known position and a mobile GPS unit receiving signals from the same satellite. We can set up two equations:

$$\begin{aligned} P_{ms} + C_m + E_{ms} &= R_{ms} \\ P_{rs} + C_r + E_{rs} &= R_{rs} \end{aligned}$$

where P_{ms} is the measured pseudorange from the mobile receive; C_m is the range error due to error in mobile receiver's clock; E_{ms} encapsulates remaining mobile error terms; R_{ms} is the actual range of the satellite from mobile receiver; P_{rs} is the measured pseudorange from the reference receive; C_r is the range error due to error in the reference receiver's clock; E_{rs} encapsulates remaining reference error terms; and R_{rs} is the actual range of the satellite from the reference receiver. Subtracting the equations to form a single difference gives:

$$(P_{ms} - P_{rs}) + (C_m - C_r) + (E_{ms} - E_{rs}) \quad = \quad R_{ms} - R_{rs}$$

Expand the expression for R_{ms} as $|\boldsymbol{X}'_m - \boldsymbol{X}_s|$ and isolate it on the RHS. $\boldsymbol{X}'_m$ is the actual position of the mobile receiver and $\boldsymbol{X}'_m = \boldsymbol{X}_m + d_{\boldsymbol{X}_m}$ where $\boldsymbol{X}_m$ is the initial estimate of the mobile receiver's position.

$$(P_{ms} - (P_{rs} - R_{rs})) + (C_m - C_r) + (E_{ms} - E_{rs}) \quad = \quad \sqrt{((\boldsymbol{X}_m + d_{\boldsymbol{X}_m}) - \boldsymbol{X}_s)^2} \qquad (3.2)$$

Compare this to the earlier equation 3.1. Consider the following correspondences:

Previous Equation 3.1	This Equation 3.2
P_{rs}	$P_{ms} - (P_{rs} - R_{rs})$
C_r	$C_m - C_r$
E_s	$E_{ms} - E_{rs}$
$\boldsymbol{X_r}$	$\boldsymbol{X}_m$
$\boldsymbol{d_X}$	$d_{\boldsymbol{X}_m}$
$\boldsymbol{X_s}$	$\boldsymbol{X}_s$

We can therefore solve for $(d_{X_m}, d_{Y_m}, d_{Z_m})$ and $C_m - C_r$ iteratively, just as we previously solved for (d_X, d_Y, d_Z) and C_r. The single differencing calculation for dual units has been reduced to the same calculation as for a single unit. This is very convenient, as there is total code reuse! The original rôle of the pseudorange is taken over by the expression $P_{ms} - (P_{rs} - R_{rs})$ which represents the mobile pseudorange corrected by the difference between the measured reference pseudorange and the actual reference range. R_{rs} is known, as the base station is at a known position. Note that the complex tropospheric and ionospheric correction terms are absent in the differential calculation, as provided the reference and mobile receivers are sufficiently close (within 20km), then these are effectively equal for both receivers and cancel in the subtraction. The pseudorange correction term is directly known and so automatically takes account of these complex terms, which thus do not have to be calculated.

3.7.1 One-Pass Iteration for Dual Receivers

Calculating the satellite positions for single differencing does not require a two-pass approach, as an adequate time of transmission (from which the satellite position is derived is) is given by:

$$T_{tr} = T_{rc} - \frac{P_{rs}}{c}$$

A reason for this is that the clock error in T_{rc} will also be reflected in the measured pseudorange, which is also dependent on the GPS receiver's clock. Thus the clock errors in T_{rc} and P_{rs} will cancel, ultimately to give a reasonable pseudorange, though admittedly one without other forms of correction. However, the technique of single differencing itself removes these other errors, so they are of no concern. (In fact this expression was also moved into the non-differential code described in the next chapter, instead of using an initial τ of 0.075, and the final solution was found to be **no** different *i.e.* a similar argument holds in this case.) An obvious guess for the mobile initial location in the iterative solution technique is the reference position. There is no need for a special external guess. Indeed, the single difference calculation is considerably simpler than the standalone calculation, despite the fact that data from two satellite receivers are being used.

3.7.2 Generalisation of Method for Type 3 Receivers

Van Leeuwen's calculation of single differences assumes that the satellites are in the same position when they transmit both the signal received by the reference and mobile receivers. While this may be true for type 1 GPS receivers, and approximately so for type 2 receivers, this cannot be assumed for type 3 receivers. The author has explored two approaches to resolve this problem.

Approach 1

Allow the satellites to assume two different sets of positions, one for each receiver. This is achieved by two standalone position fixes. For each receiver, the expression:

$$T_{tr} = T_{rc} - \frac{P_{rs}}{c}$$

is still used in the differential calculation to establish transmission time (and hence satellite positions) for each receiver because it eliminates clock error. It would be possible to use:

$$T_{tr} = T'_{rc} - \frac{P'_{rs}}{c}$$

where T'_{rc} and P'_{rs} are the corrected unit clock and pseudoranges that provide a best-fit standalone solution. Instead we remove the complication of pseudorange correction terms by adopting the former expression — we do not want to lose the advantage in single differencing of ignoring these corrections. However, the value of τ used to calculate the rotation of the Earth has been calculated from the corrected pseudorange (*i.e.* P'_{rs}/c) established in the standalone calculation because it will be more accurate than P_{rs}/c . The calculation goes in two phases:

1. determine pseudorange corrections with each satellite assumed to be in the same position for both receivers
2. apply pseudorange corrections with satellites in the per-receiver calculated positions

Approach 2

A second method is to interpolate both the pseudorange and carrier phase from both receivers to the nearest GPS second. The time and range measurement modifications are shown:

$$\begin{aligned} T'_{n+1} &= \lfloor T_{n+1} + 0.5 \rfloor \\ R'_{n+1} &= R_{n+1} + (T'_{n+1} - T_{n+1}) \times \frac{(R_{n+1} - R_n)}{(T_{n+1} - T_n)} \end{aligned}$$

Which Method is Better?

Method 1 was chosen as the more general and flexible solution, although method 2 was also coded and used. Method 2 is used practically in some interpolation code to correct data files from Type 3 receivers for subsequent processing, but it assumes an unwarranted linearity. Also, performing two standalone position calculations to establish satellite position initially is more reliable than the basic single difference approach, because the least squares method used reduces errors in these satellite positions. Overall, the author's enhancement (Method 1) would be expected to be the more reliable, and this was found to be the case in practice.

GPS Pseudorandom Sequence		
name	rate	period
P-Code	1023 kbs	1 ms
C/A	10.23 Mbps	7 days

Table 3.1: GPS sequences

3.7.3 Smoothing of Pseudorange by Carrier Phase

The pseudorange is calculated from the content of the radio signal, and may be called a code-phase measurement. The signal is a sequence of pseudorandom numbers which repeats. The signal that is received is correlated against the same pseudorandom sequence synthesised by the GPS unit until a fit is obtained. In fact two sequences are transmitted (see Table 3.1), a slower one of higher frequency (called the Coarse Acquisition or C/A-code) to establish a number of candidate coarse correlations and a faster one of lower frequency (called the P-code) to distinguish between these to provide an absolute correlation. The largest possible error in the pseudorange will be half a period. For the C/A code this is $0.5 \times \frac{c}{10^6} \approx 150$ m. However, a reasonable receiver (of consumer grade) can correlate to around 1% of this period, so the level of measurement error in the pseudorange is about 3 metres. Is it possible to obtain greater accuracy?

In addition to the pseudorange, some GPS units are also capable of measuring the phase of the radio signal carrier. This is called a carrier phase measurement, and a reasonable receiver can again measure phase to around 1% of a carrier wave. As the wavelength is around 19 cm, this represents a high accuracy of 2 millimetres. However, the code phase $0 \leq p < 2\pi$ is a relative rather than absolute measure, and all one knows is that a more accurate measurement of the pseudorange is $(N + \frac{p}{2\pi})\lambda, N \in \mathbb{Z}$. Most receivers will try to keep a relative track of N, to provide less ambiguity. This is called the integrated carrier phase and it is presented as $(2\pi N' + p), N' \in \mathbb{Z}$, where $(N' - N)$ is constant. In practice so-called "cycle slips" will occur from time to time and the value of $(N' - N)$ will jump, unbeknownst to the unit. Clearly, this degree of cycle-slipping depends on the quality of the receiver and the reception conditions, but cycle slips are a fact of life and must be modelled.

By the way of irony, for the sake of simplicity in the next derivation, assume that cycle slips do not happen! The basic problem is to combine a less accurate but absolute value of a pseudorange, with a more accurate but relative measure of the same pseudorange. The intention is to obtain a pseudorange which is absolute and is highly accurate. Consider, a corresponding sequence of pseudoranges and

carrier ranges: $P_1, \ldots P_n$, and $C_1, \ldots C_n$ respectively:

$$\begin{aligned} C_n - C_i &= P_n - P_i \quad 1 \le i \le n \\ P_n &= P_i + C_n - C_i \quad 1 \le i \le n \end{aligned}$$

The equation above represents n different expressions for P_n. The average of these expression should be more precise than the observation P_n itself.

$$\begin{aligned} \overline{P_n} &= \frac{1}{n}\sum_{i=1}^{n}(P_i + C_n - C_i) \\ &= \frac{1}{n}\sum_{i=1}^{n} P_i + C_n - \frac{1}{n}\sum_{i=1}^{n} C_i \\ &= \frac{1}{n}\left(\sum_{i=1}^{n-1} P_i + P_n\right) + C_n - \frac{1}{n}\left(\sum_{i=1}^{n} C_i + C_n\right) \\ &= \frac{n-1}{n}\left(\frac{1}{n-1}\sum_{i=1}^{n-1} P_i + C_{n-1} - \frac{1}{n-1}\sum_{i=1}^{n-1} C_i\right) + \frac{1}{n}P_n - \frac{n-1}{n}C_n + C_n + \frac{1}{n}(P_n - C_n) \\ &= \frac{n-1}{n}\overline{P_{n-1}} + \frac{1}{n}P_n - \frac{n-1}{n}C_{n-1} + \frac{n-1}{n}C_n \\ &= \frac{1}{n}\left(P_n + (n-1)\left(\overline{P_{n-1}} - C_{n-1} + C_n\right)\right) \end{aligned}$$

We have arrived at a convenient recursive form for calculating the smoothed pseudorange. Theoretically, the more epochs involved in the calculation, the more accurate will be the result. However, the effect of the ionosphere is to advance the carrier range but to reduce the pseudorange. Hence C_n and P_n do not quite maintain a constant difference if conditions vary — the effect is called "ionospheric divergence". The other problem is a possible cycle-slip in the integrated carrier range which will happen if the lock on the satellite is lost **or** the rate of change is too great. To overcome these two problems, only a limited number of observations are used in smoothing, 100 being a good compromise. Large cycle slips can be detected as $(P_n - P_{n-1}) - (C_n - C_{n-1}) > 15$ m, say, and the integration period is started from scratch. The benefit of smoothing for that particular pseudorange will be lost for a time. Small undetected cycle slips may still occur, but their impact is less significant. The AllStar GPS unit used actually sets a bit to indicate a cycle slip, and this has been observed

to happen infrequently. The documentation does not reveal whether this is baseline truth, or merely surmised by a criterion such as the above. The Garmin G12XL unit appears to cycle-slip relatively frequently (using the above test) in comparison. The value of the threshold for "detecting" a cycle slip depends on the quality of the receiver. For high quality receivers and optimally located antennae this can be as low as 1 m. For the Garmin G12XL, the optimal value for this threshold was found to be 2.6 m (see the next chapter).

All the pseudorange calculations above can be used with smoothed pseudorange figures, though only the single differencing method (rather than the standalone method) is likely to benefit from the increased accuracy. In single differencing, the reference receiver set-up makes the connection between a relative measure and an absolute known distance.

3.8 Sources of GPS Data

The GPS system is a complex one that only exists due to a vast organisational infrastructure. Any positional calculation is only as good as the measurements taken and the parameters provided by that infrastructure. While measurements can be taken only once and represent a "ground truth", the parameters supplied can be of varying quality. For example, when post-processing GPS data it is usual to be able to use a set of orbital parameters which are more accurate than those which could have been obtained at the time of the observations. Therefore, it is important to pay attention to the routes by which input parameters to a GPS calculations are obtained.

3.8.1 GPS Receivers

The majority of GPS units speak a standard RS232-like protocol called NMEA [49]. This is simple-minded and contains only basic and generally device-independent navigation information such as a position fix. The NMEA protocol does not support the transmission of the requisite data to compute a GPS position fix, despite the fact that the receiver has undoubtedly received and decoded this information from the satellite signals. In short, the so-called "raw data" are in there but there is no standard way to get at them. A few manufacturers of more specialised equipment support a proprietary protocol; the majority of manufacturers keep any such protocol (if it indeed exists) secret.

The protocol by which differential corrections are supplied is called RTCM [47] and is similarly incapable of communicating the necessary data for a position calculation. All that RTCM consists of

is a set of range corrections. Indeed, though most GPS units can understand RTCM, only a few much more highly priced dedicated units can actually produce RTCM.

The reticence of the majority of GPS receivers to reveal the information they are receiving from the satellites and their low level measurements makes the choice of equipment very small indeed if one wants to perform one's own GPS calculations.

3.8.2 GPS Data on The Internet

The National Geodetic Survey (NGS) of America maintain a network of Continuously Operating Reference Stations. Archived data are available on:

`http://www.ngs.noaa.gov/CORS/Data.htm`

Similarly, the International GPS Service (IGS) operates a global network of 100 dual frequency GPS stations and provides archived data on:

`http://igscb.jpl.nasa.gov/`

Note that in the case of ephemeris parameters, three versions are available: predicted, rapid and final (in increasing order of quality). The rapid and final combinations are available with 2-day and 2-week latencies respectively (due to the necessary global calculations), whereas the predicted orbital parameters are available for a single day, being made available at about 23:00 UT on the previous day. The corresponding orbit quality is given in the following table:

ephemeris name	latency	orbit accuracy
final	2 weeks	better than 5 cm
rapid	2 days	slightly worse than 5 cm
predicted	real-time	10 – 50 cm

The ephemeris data that are broadcast by the satellites themselves are necessarily a form of predicted ephemeris, so a real-time position fix can never be as accurate as a post-processed fix. On the other hand, the current organisation of GPS data on the Internet is still very much anachronistically batch-processed. With increasingly powerful CPUs, there is scope for services to be near real-time. Instead of explicitly downloading a file of appropriate vintage, it should be possible to connect "live" to a correction service that will transparently return the correction data for the present moment or for historic periods as appropriate.

The presence of predictive ephemeris data on the net **is** an alternative source to a GPS unit for this type of information, so the very minimum "raw data" required from a GPS unit are a time and corresponding pseudorange measurements. In theory, the data predicated up to 24 hours ahead should not be as good quality as the real-time predictive data broadcast by the satellites, but in practice results would depend very much on their respective qualities of implementation.

3.9 High Accuracy GPS

High accuracy double difference GPS methods require resolving the ambiguities in the double difference carrier phase data. This resolution involves finding a set of integers which when multiplied by the signal wavelength, most self-consistently describe the set of experimentally measured double difference carrier phase ranges. "Self-consistently" means best fitting the measured data as well as best satisfying the geometric constraint that the measurements were made at a single point.

The standard solution without resolving the ambiguities is known as the "float solution". If the integer ambiguities can be successfully resolved then a so-called "fixed solution" has been found. The fixed solution will be considerably more accurate than the float solution. A number of different techniques for determining the ambiguities exist. The most promising located is the LAMBDA (Least-squares AMBiguity Decorrelation) method [50]. Two public domain LAMBDA codes are identifiable: one originating from the University of Delft [51] and the other from a GPS textbook "Linear Algebra, Geodesy, and GPS"[45]. These are examined in the next chapter.

The LAMBDA method has been developed since 1993 at the Delft University of Technology to resolve integer ambiguities. The advantage and novelty of the method is a decorrelating reparameterisation of the ambiguities, which allows an integer least squares estimation to be computed very quickly *i.e.* in a couple of milliseconds using current processor technology. The search space for integer solutions is a hyper-ellipsoid that is extremely elongated. By reparameterising the ambiguities the hyper-ellipsoid can be transformed into a near spheroid. Note that the number of candidate solutions is the same, but that they can be found much more efficiently.

Chapter 4

GPS for Augmented Reality

4.1 System Choices

The Global Positioning System (GPS) described in the previous chapter is an obvious candidate as a positioning technology for the outdoors Gosbecks Archælogical Park Augmented Reality application. This chapter reports on the practical experience of constructing GPS systems and the results from the various solution developed. This section looks at the choice of GPS systems available, and how the requirements of the AR application guided our selection.

4.1.1 Subscription *vs.* DIY Differential GPS

A clear choice has to be made between subscribing to a differential GPS correction service, and setting up one's own base station with some form of (radio) link. Differential service subscription costs are high: of the order of £1000 a year for satellite-sourced corrections and £500 for FM radio-sourced corrections. Note that the satellite which supplies the differential GPS correction service is **not** a GPS satellite but one in geostationary orbit (unlike the GPS satellites). The FM correction service within the UK is broadcast within the digital Radio Data System (RDS) component of the Classic FM station, which has nationwide coverage. Even the cheapest differential receiver (for the free maritime beacon service) is of the order of $450.

Criticisms are levelled against DIY DGPS, because unless a large enough common subset of satellites are within range at both locations and sufficiently high quality information can be obtained, results have been reported as worse than with a single receiver. On the other hand, the error of differential GPS bears a linear relationship to the length of the baseline, if this is below around 30 km.

System Option	Description	Cost	Performance
1	DIY base station with radio link to GPS receiver	£11,995	50 cm @ 1 Hz
2	satellite differential service	£8,285 (includes 1^{st} year subscription)	1 m @ 1 Hz
3	maritime beacon receiver with radio link to GPS receiver	£8,213	1 m @ 10 Hz
4	DIY dual frequency base station with radio link to dual frequency receiver	£28,881	1 cm @ 1 Hz
5	FM differential service	£1,393 (includes 1^{st} year subscription)	$<$ 5m @ 1 Hz using GPS 35 1-5m @ 1 Hz using GPS 12XL

Table 4.1: DGPS system quotes

DIY DGPS allows the length of the baseline to be controlled accurately; subscription users are at the mercy of their service provider's nearest base station, and indeed the longevity of that service.

4.1.2 Hardware

Quotes were obtained for a variety of bespoke differential GPS system in 1998, see Table 4.1. All were extremely expensive and were consequently unaffordable. In spite of the cost, the accuracy claimed by these systems, except the most expensive at around £30,000, was only of the order of a metre, barely in the ballpark for AR.

The nearest maritime beacon to Colchester was identified at North Forland in Margate. The service had only been in operation a week when located, but at 40 miles from Colchester, the North Forland beacon is not going to give an ideal correction signal.

The highest end differential GPS systems are capable of phenomenal performance: figures are given of 1mm accuracy at 10 Hz, but these are only available at phenomenal prices. High-end GPS is used in surveying and niche maritime applications, such as docking supertankers to millimetre accuracy. One must be wary, however, whether accuracy figures quoted are post-processed or achievable real-time. AR requires real-time accuracy.

To obtain high accuracy much processing power is required, and it is only relatively recently that "real-time kinematic" techniques have provided high levels of real-time accuracy. More sophisticated forms of differential GPS are employed — instead of the basic single differencing computations, double or even triple differencing may be used, as described in the previous chapter. The downside of these more sophisticated processing techniques is that they can place considerable constraints on the set-up — often requiring long initialisation runs; high quality hardware and ideal reception conditions. Because real-time kinematic differencing techniques work on a probabilistic basis when they go wrong they go spectacularly wrong. What level of demand can we place upon the typical user of an AR system that is based on a wearable computer?

It is clear that manufacturers of GPS equipment and the providers of differential GPS correction services are putting a considerable price premium on high accuracy. A single handheld GPS unit retails for $100. A rudimentary differential system should cost little more than twice this amount, yet customers are being charged thousands of dollars.

The market is changing, however, at least at the intermediate accuracy level. With GPS functionality increasingly being integrated upon ever more low power ASICs and with SA switched off, the capability of knowing one's location to within a few metres will soon become omnipresent. This is a significant quantitative change. Hitherto such accuracies were only obtainable from expensive differential GPS systems. Phones and watches are now available with embedded GPS.

An open question is whether, with SA switched off, there is any significant additional advantage in differential GPS except at the very high end of the market. AR requires the accuracy only currently available by differential techniques, even after the switch off of SA, but can this be achieved using inexpensive hardware? To stop the public building their own differential GPS systems at low cost, manufacturers have taken the following steps;

- General GPS hardware has been built only to accept a differential GPS correction signal, but **not** to generate one.
- Hardware that generates a differential GPS signal (and consider that this is merely a firmware change) puts the price up considerably *i.e.* by thousands of dollars!
- It is impossible to extract the information recovered by a typical GPS unit, broadcast by the satellites, that is used in a GPS position calculation. If this information could be obtained from a single receiver, it would be possible to produce a DIY stream of RTCM differential correction

data. Further, if this information could be obtained from a couple of receivers, it would be possible to do DIY differential position calculations, circumventing the RTCM correction protocol entirely.

It is probable that for testing purposes *etc.* , most GPS manufacturers support a communications protocol that allows the extraction of all the low-level satellite information from each GPS receiver. However, these protocols are proprietary, undocumented and secret. There are no protocol standards defined for this type of information. At the stage that equipment was being purchased, none of these secret protocols had been fully "cracked" by the Internet community, only sketchy information existed. However, towards the end of the project, there was a major breakthrough in extracting the most significant information from the inexpensive and most widespread handheld GPS receiver: the Garmin G12XL. Again, this radically changed the playing field, and the details are discussed in Section 4.7.

There are, however, a small number of relatively inexpensive OEM products that make the low-level information available via proprietary, documented and supported protocols. A centralised list of this equipment is maintained by Sam Storm Van Lewven [48], who is the *doyen* of the DIY GPS Internet community. Development based on a supported product should in theory be a more reliable approach as there is recourse to the manufacturer. Working with low-level data demands that we perform the GPS position calculations ourselves instead of relying on the firmware of the GPS unit. However, the subsidiary benefit of using homebrew code is the complete control over the system architecture and the ability to tune the code to any particular application to achieve greater accuracy.

In the end we decided to go for a DIY differential GPS system based upon two AllStar [52] boards (made by CMC in Canada) which are typical of the latest OEM products that provide phase information, but only cost a few hundred dollars. The model option we chose supplies raw data at 10 Hz which enables our system to achieve near real-time responsiveness. Most GPS units provide a position fix once a second at most.

To summarise, we are spared considerable hardware expense by performing the differential position calculation ourselves using homebrew GPS code. It is expensive to buy compute power in a GPS product (a 10Hz position fix model is considerably more expensive than a 10Hz raw data model), but cheap to buy compute power in a PC.

4.2 Software

4.2.1 Device Independent

The starting point was some public domain GPS software, written by Sam Storm van Leuwen. This remained for some time the only readily accessible open source software for calculating GPS position fixes. Subsequently some equivalent MATLAB code was discovered and used (see Section 4.16.3). Van Leuwen's software is written in Pascal and consists of three executables:

1. **stdalone**: standalone (*i.e.* single receiver) position calculation
2. **singldif**: single difference, dual receiver position calculation
3. **sdsmooth**: single difference, dual receiver position calculation with carrier phase smoothing

These programs are batch processing: reading input files and producing output files. There is a standard file format for some types of GPS data called RINEX. Unfortunately, the requirements of DIY GPS processing are not fully met by this format and so van Leuwen's files adhere to his own format! Code for an executable called "Ridinter" to perform input file manipulation to make "sdsmooth" work with a Type 3 receiver (receiver types are defined in Chapter 3) was subsequently received from van Leuwen. An updated version of "stdalone" called "Zd", with some enhancements was also received. All Pascal code was ported to 'C' and mounted on the Web [1]. Many people have since used this as the basis for the development of their own GPS code.

4.2.2 Device Dependent

The software supplied for the AllStar receiver was Windows-based. It consists of the following executables:

1. **gpsmon**: general and powerful monitor utility to communicate to GPS unit
2. **prog**: programming utility to update firmware
3. **gps2txt**: decodes limited basic information from GPS unit and writes this to a file

Only the 'C' source of "gps2txt" was supplied, and aside from this starting point, the software to talk to the AllStar unit under Linux had to be written from scratch. Naturally, the source of the AllStar

[1] `http://privatewww.essex.ac.uk/~djjohn`

software was requested — no response arrived from Marconi. It became clear early that "gps2txt" only supports a very limited number of the unit → host message types, and because of its nature none of the host → unit message types. Tables 4.2 and 4.3 show the proprietary message types that are exchanged between the host CPU and the AllStar GPS unit. The † indicates that the message type is supported in the software that has been written: 34 out of the 60 message types have been supported. Only 6 out of these 34 (annotated with ‡ in the tables) were supported in the supplied open source Marconi software. Basically all message types were supported which could be possibly implicated in a differential GPS system.

4.2.3 Software Integration

This was performed in three stages:

1. Generate a separate executable that reads data from a GPS receiver (via an RS232 port) and which writes to van Leuwen-format input files.
2. Test the combination of the RS232 → file software and the batch processing software, for valid positional output.
3. Integrate the two executables into a single executable, that avoids writing the intermediate input file. This is the "real-time" program.

The most difficult part was stage 1 which required identifying corresponding GPS quantities across 6 different manifestations given in Table 4.4. Table 4.4 points to the appropriate entries in tables 4.5 and 4.6 which present the mappings established. These mapping tables represent the data flow stages of a DIY GPS position fix, and are the key behind the interfacing of hardware and software. Problematically, the atmospheric parameters a_0, a_1, a_2, a_3, b_0, b_1, b_2 and b_3 to calculate the ionospheric delay are **not** provided by the AllStar receiver, despite the fact that they are present in the satellite signal and must have been decoded by the unit. Marconi was contacted to determine if the atmospheric parameters could be extracted by whatever means: no reply was forthcoming. A clear recommendation that emerges from this work is that all GPS units should simply release all the "raw data" in precisely the same format that it is broadcast by the satellite. Not only would there be no missing raw data, but software to interpret this data would work with equipment from all manufacturers. There is no reason to invent an additional protocol to the one defined in ICD-GPS-200C [53] describing the satellite signals.

Host CPU to Receiver Messages † = DIY implemented		
ID	Description	Name in Code
†6	Current channel assignment data request	CURRENT_CHANNEL_ASSIGNMENT_DATA_REQUEST
†13	Request raw data for channels	SECRET_MEASUREMENT_BLOCK_DATA_REQUEST
†20	Navigation data request (user coordinates)	NAVIGATION_DATA_REQUEST_USER_COORDINATES
†21	Navigation data request (GPS coordinates)	NAVIGATION_DATA_REQUEST_GPS_COORDINATES
†22	Ephemeris (ICD-GPS-200 format) request	EPHEMERIS_REQUEST
†23	Measurement block data request	MEASUREMENT_BLOCK_DATA_REQUEST
†33	Satellite visibility data and status request	SATELLITE_VISIBILITY_DATA_AND_STATUS_REQUEST
†43	DGPS Confi guration request	DGPS_CONFIGURATION_REQUEST
†45	Hardware/Software identifi cation	SOFTWARE_IDENTIFICATION_INFORMATION_REQUEST
†47	Base Station Status request (optional)	
†48	Differential Message Status request	DIFFERENTIAL_MESSAGE_STATUS_REQUEST
49	Receiver Status request	
50	Satellite health summary request	
51	Initiated BIT request	
†63	Initiate link	INITIATE_LINK
64	Set Channel deselection	
65	Raw DGPS Data Request (optional)	
77	Update almanac	
78	Common almanac data transfer	
79	Specifi c almanac data transfer	
†80	Set User's Position/Operating Mode	SET_OPERATING_MODE
81	Set Mask angle	
82	Transmit DGPS data message	
†83	Set DGPS Confi guration	SET_GPS_CONFIGURATION
84	Set tropo model use	
85	Set Beacon Receiver Status	
86	Set Mean Sea Level model use	
†88	Select/Defi ne datum to use	SELECT_DEFINE_DATUM
90	Set SV deselection	
†91	Differential Message Confi guration (optional)	SET_DIFFERENTIAL_MESSAGE_CONFIGURATION
95	Track SV request	
†99	Erase NVM	ERASE_MEMORY
†103	Set Date,Time & GPS Time Alignment Mode	SET_TIME_ALIGNMENT
105	Set default CMC Binary message list	
†110	Confi gure Main Port Mode	CONFIGURE_MAIN_PORT_MODE
112	Switch to Reprogramming Mode	

Table 4.2: AllStar message types — host to receiver

Receiver to Host CPU Messages † = DIY implemented ‡ = CMC implemented		
ID	Description	Name in Code
‡6	Current channel assignment data (1-6)	CURRENT_CHANNEL_ASSIGNMENT_DATA_1_6
‡7	Current channel assignment data (7-12)	CURRENT_CHANNEL_ASSIGNMENT_DATA_7_12
†13	Raw data for channels 1 to 3	SECRET_MEASUREMENT_BLOCK_DATA_1_3
†14	Raw data for channels 4 to 6	SECRET_MEASUREMENT_BLOCK_DATA_4_6
†15	Raw data for channels 7 to 9	SECRET_MEASUREMENT_BLOCK_DATA_7_9
†16	Raw data for channels 10 to 12	SECRET_MEASUREMENT_BLOCK_DATA_10_12
‡20	Navigation data (user coordinates)	NAVIGATION_DATA_USER_COORDINATES
‡21	Navigation data (GPS coordinates)	NAVIGATION_DATA_GPS_COORDINATES
‡22	Ephemeris (ICD-GPS-200 format) data	EPHEMERIS_DATA
‡23	Measurement block data	MEASUREMENT_BLOCK_DATA
†33	Satellite visibility data and status	SATELLITE_VISIBILITY_DATA_AND_STATUS
†43	DGPS Confi guration	DGPS_CONFIGURATION
†45	Hardware/Software identifi cation request	SOFTWARE_IDENTIFICATION_INFORMATION
47	Base Station Status data (optional)	
†48	Differential Message Status request	DIFFERENTIAL_MESSAGE_STATUS
49	Receiver Status request	
50	Satellite health summary	
51	Initiated BIT result	
†63	Initiate link	INITIATE_LINK
78	Almanac reception status	
83	RTCM data Message Retransmission	
85	Beacon Receiver Status	
125	Link overload error message	
†126	Acknowledge message	ACKNOWLEDGE_MESSAGE

Table 4.3: AllStar message types — receiver to host

Manif^n No	Description	Position in Mapping Tables 4.5 and 4.6	
		Column Name	Column No
1	fields in satellite messages	Location in Satellite Data	6
2	variables in GPS standard document (ICD-GPS-200C)	Symbol	1
3	messages in AllStar unit's protocol	AllStar Message Type	5
4	variables in AllStar software	AllStar I/O S/W Structure Element	2
		AllStar I/O S/W Structure Type	4
5	data items in van Leuwen-format file	Computation Kernel Variable (array on RHS gives ordering in file)	3 (RHS)
6	variables in van Leuwen's GPS code	Computation Kernel Variable (name is on LHS)	3 (LHS)

Table 4.4: Manifestations of GPS quantities

Does the absence of ionospheric parameters exclude us from doing a DIY GPS calculation? No, the differential GPS calculation does not use the atmospheric parameters and we intend always to operate in differential mode to obtain the necessary accuracy. For a standalone GPS calculation there there are two possibilities. Firstly, all the atmospheric parameters can be taken as zero which is equivalent to ignoring the ionospheric correction term. Secondly, the atmosphere parameter values could be always taken as those in the test input data file. Perhaps the use of **any** atmospheric parameter values for correction will be better than using none?

The supplied documentation was poor (particularly for the AllStar unit) so values had to be examined at run time to "guess" their identity — a task further complicated by the use of inconsistent units. There was considerable relief when the first believable position fix was obtained. The resulting homebrew GPS software is called rather descriptively "diygps". This incorporates all the functionality of van Leuwen's code. Command line switches are used to choose between standalone, differential and carrier phase smoothed differential operation.

4.3 Initial Experiments

Two GPS antennae were positioned on the roof of the university, adjacent to one another so that they were essentially in the same position. This is called a "zero-baseline" set-up and is the most sensitive way to detect noise and error in the system. The view of the sky in this location was partially restricted with a vertical concrete wall of an additional storey about 15 feet from the antennae. Typically around 6 satellites were visible. A mark was made on the roof, and the position was surveyed by averaging the GPS position over a long time period.

4.3.1 DIY *vs.* Firmware

The positions reported by the GPS unit were compared with those produced by our DIY software. The loci of positions reported by both are shown in Figure 4.1. These can be seen to follow each other closely, verifying the correctness of the DIY software. There is a small difference, of the order of a few centimetres generally, which separates the positions reported at the same time. 2311 readings at one-second intervals were taken. The ionospheric parameters are set to zero. The error distributions in Figure 4.2 shown a modal error of just under 2m with a rapidly diminishing tail-off.

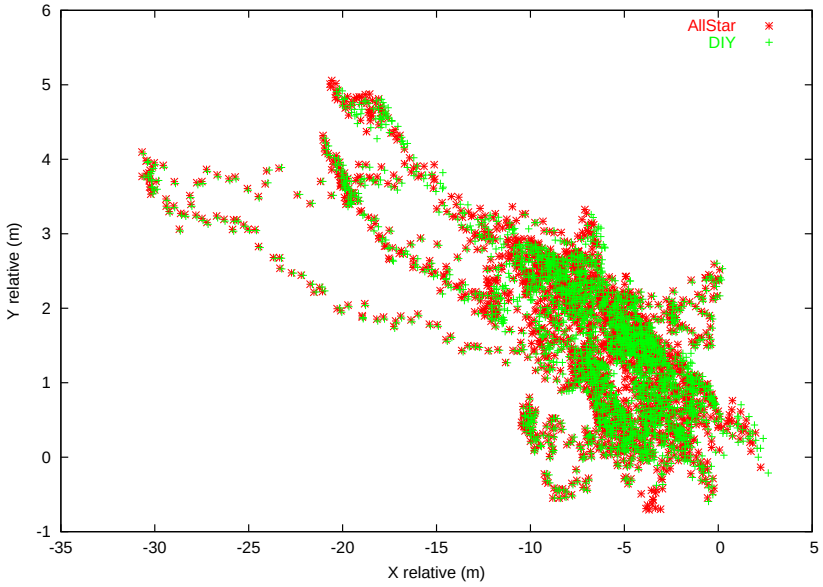

Figure 4.1: DIY *vs.* firmware position plots

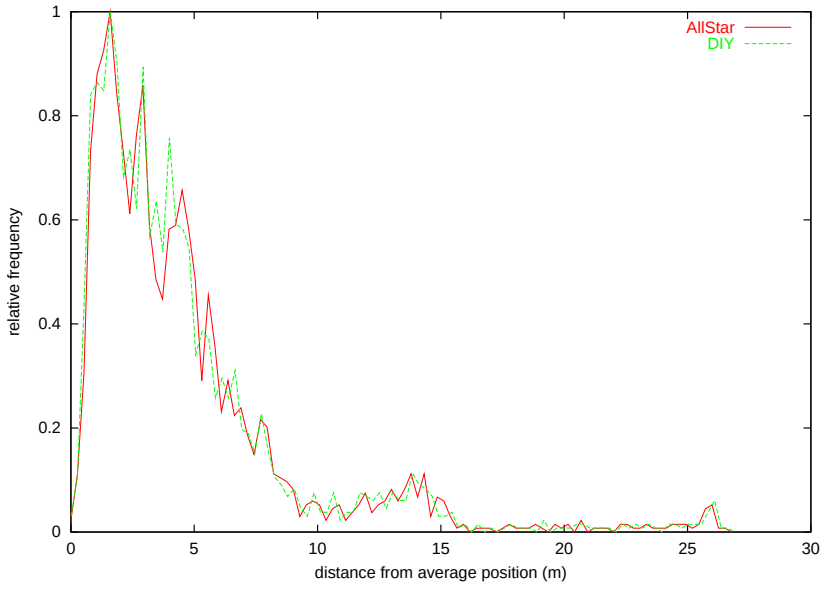

Figure 4.2: DIY *vs.* firmware error distribution

Per Satellite Data					
Symbol	AllStar I/O S/W Structure Element	Computational Kernel Variable (arranged as batch fi le input)	AllStar I/O S/W Structure Type	All-Star Msg Type	Subframe Location in Satellite Data
P_{rs}	`Code_Phase`	`SV[sv] or P[sv]= (Code_Phase/1023000*2048) * 299792458.0;`	FAST_MEAS_BLOCK_TYPE	23	N/A
C_r	`ICP`	`C[sv]= (((ICP & 0xfffff000)>>12) +((ICP&0xfff)>>2)/1024.0)) * 0.190293672`	FAST_MEAS_BLOCK_TYPE	23	N/A
C_{rs}	`Crs`	`eph[sv][ 0]= Crs;`	ORBIT_PARAMETERS_TYPE	22	2
Δn	`del_N`	`eph[sv][ 1]= del_N/PI;`	ORBIT_PARAMETERS_TYPE	22	2
M_0	`M0`	`eph[sv][ 2]= M0/PI;`	ORBIT_PARAMETERS_TYPE	22	2
C_{uc}	`Cuc`	`eph[sv][ 3]= Cuc;`	ORBIT_PARAMETERS_TYPE	22	2
e	`E`	`eph[sv][ 4]= E;`	ORBIT_PARAMETERS_TYPE	22	2
C_{us}	`Cus`	`eph[sv][ 5]= Cus;`	ORBIT_PARAMETERS_TYPE	22	2
$A^{\frac{1}{2}}$	`Root_A`	`eph[sv][ 6]= Root_A;`	ORBIT_PARAMETERS_TYPE	22	2
t_{oe}	`Toe`	`eph[sv][ 7]= Toe;`	ORBIT_PARAMETERS_TYPE	22	2
C_{ic}	`Cic`	`eph[sv][ 8]= Cic;`	ORBIT_PARAMETERS_TYPE	22	3
Ω_0	`Omega_0`	`eph[sv][ 9]= Omega_0/PI;`	ORBIT_PARAMETERS_TYPE	22	3
C_{is}	`Cis`	`eph[sv][10]= Cis;`	ORBIT_PARAMETERS_TYPE	22	3
i_0	`I_0`	`eph[sv][11]= I_0/PI;`	ORBIT_PARAMETERS_TYPE	22	3
C_{rc}	`Crc`	`eph[sv][12]= Crc;`	ORBIT_PARAMETERS_TYPE	22	3
ω	`W`	`eph[sv][13]= W / PI;`	ORBIT_PARAMETERS_TYPE	22	3
$\dot{\Omega}$	`Omega_Dot`	`eph[sv][14]= Omega_Dot/PI;`	ORBIT_PARAMETERS_TYPE	22	3
$\dot{i}$	`I_Dot`	`eph[sv][15]= I_Dot/PI;`	ORBIT_PARAMETERS_TYPE	22	3
T_{gd}	`Tgd`	`clk[sv][ 0]= Tgd;`	CLOCK_PARAMETERS_TYPE	22	1
T_{oc}	`Toc`	`clk[sv][ 1]= Toc;`	CLOCK_PARAMETERS_TYPE	22	1
a_{f2}	`Af2`	`clk[sv][ 2]= Af2;`	CLOCK_PARAMETERS_TYPE	22	1
a_{f1}	`Af1`	`clk[sv][ 3]= Af1;`	CLOCK_PARAMETERS_TYPE	22	1
a_{f0}	`Af0`	`clk[sv][ 4]= Af0;`	CLOCK_PARAMETERS_TYPE	22	1

Table 4.5: Link between satellite H/W and S/W — per satellite data

4.3.2 DIY $\pm$ Ionospheric Parameters

To investigate the effect of the ionospheric parameters, the DIY position calculation was performed twice both using the sample ionospheric parameters in the test input data, and again by setting all these parameters to zero. 3014 readings at one-second intervals were taken. The position plots are shown in Figure 4.3. The error distributions are shown in Figure 4.4. It is clear that the ionospheric parameters **do** make a difference, though a relatively small one as the positions follow each other to within a few centimetres.

Per Position Calculation Data					
Symbol	AllStar I/O S/W Structure Element	Computational Kernel Variable (arranged as batch fi le input)	AllStar I/O S/W Structure Type	All-Star Msg Type	Subframe Location in Satellite Data
T_{rc}	GPS_Time	`Trc`	FAST_MEAS_BLOCK_TYPE	23	N/A
a_0	N/A	`a0 = ion[0];`	N/A	N/A	4
a_1	N/A	`a1 = ion[1];`	N/A	N/A	4
a_2	N/A	`a2 = ion[2];`	N/A	N/A	4
a_3	N/A	`a3 = ion[3];`	N/A	N/A	4
b_0	N/A	`b0 = ion[4];`	N/A	N/A	4
b_1	N/A	`b1 = ion[5];`	N/A	N/A	4
b_2	N/A	`b2 = ion[6];`	N/A	N/A	4
b_3	N/A	`b3 = ion[7];`	N/A	N/A	4

Table 4.6: Link between satellite H/W and S/W — per position calculation data

Set Up	Average Position	Maximum Error	Average Error
Allstar	$X = 3945317.0 \pm 0.2$ m $Y = 65224.20 \pm 0.05$ m $Z = 4994361.1 \pm 0.1$ m	26.52 m	9.62 m
DIY Allstar	$X = 3945318.2 \pm 0.2$ m $Y = 65224.17 \pm 0.05$ m $Z = 4994361.2 \pm 0.1$ m	26.63 m	9.51 m

Table 4.7: Firmware *vs.* DIY position

Set Up	Average Position	Maximum Error	Average Error
- ionic	$X = 3945327.37 \pm 0.07$ m $Y = 65224.25 \pm 0.03$ m $Z = 4994372.99 \pm 0.06$ m	7.52 m	4.71 m
+ ionic	$X = 3945327.30 \pm 0.07$ m $Y = 65224.48 \pm 0.03$ m $Z = 4994371.71 \pm 0.07$ m	7.77 m	4.87 m

Table 4.8: Effect of ionospheric parameters on position

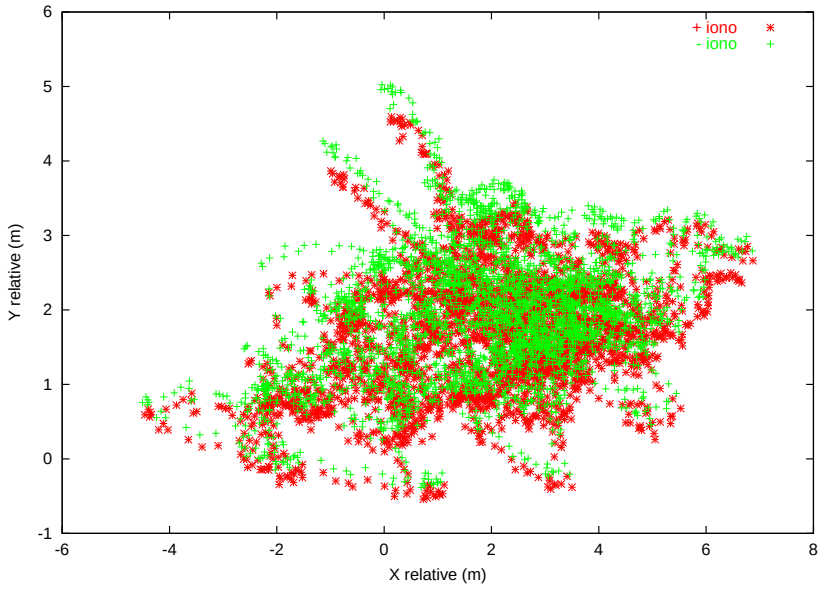

Figure 4.3: DIY ± ionospheric parameters: position plots

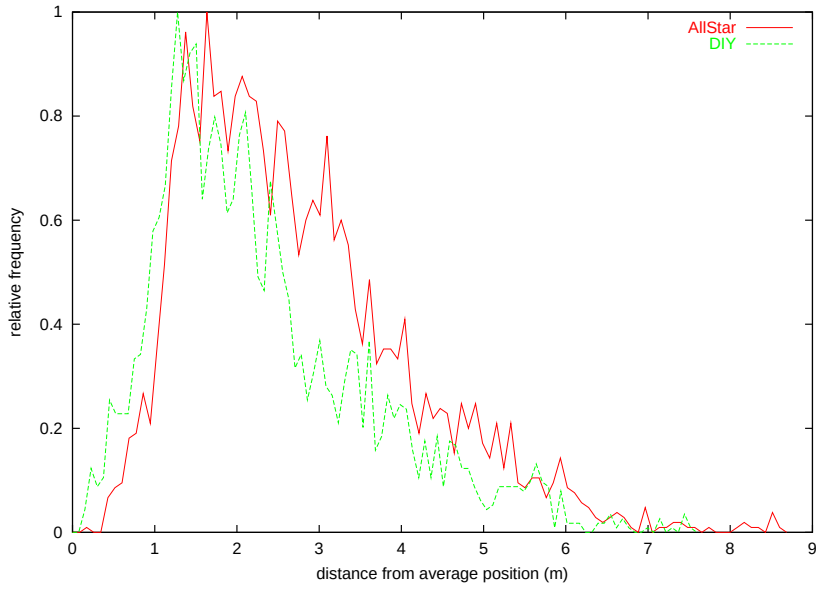

Figure 4.4: DIY ± ionospheric parameters: error distributions

setup	position	range
single receiver (mobile)	$X = 3945635.33 \pm 2.55$ m $Y = 65212.29 \pm 0.45$ m $Z = 4994441.98 \pm 1.81$ m	$DX = 236.54$ m $DY = 27.80$ m $DZ = 86.44$ m
single receiver (reference)	$X = 3945752.06 \pm 2.00$ m $Y = 65179.20 \pm 0.57$ m $Z = 4994416.69 \pm 0.82$ m	$DX = 55.30$ m $DY = 16.12$ m $DZ = 23.56$ m
simple differential GPS	$X = 3945593.16 \pm 0.67$ m $Y = 65496.18 \pm 0.26$ m $Z = 4994168.31 \pm 0.14$ m	$DX = 16.12$ m $DY = 5.85$ m $DZ = 4.34$ m
phase differential GPS	$X = 3945590.81 \pm 1.94$ m $Y = 65496.28 \pm 0.32$ m $Z = 4994168.14 \pm 0.29$ m	$DX = 94.59$ m $DY = 13.28$ m $DZ = 19.96$ m

Table 4.9: GPS positions from varying techniques (200 measurements, SA on)

4.3.3 DIY Differential GPS

The initial results for differential GPS, obtained while SA was switched on, shown in Table 4.9 looked relatively encouraging. Differential GPS improves accuracy, but phase differential GPS is unexpectedly worse than simple differential GPS. This anomaly was eventually traced to a hardware fault and the next section describes the hardware problems which hindered further progress.

4.4 Hardware Problems

Initial trials using two AllStar GPS receivers, showed simple differential GPS gives some improvement. However, phase differential GPS does not provide the expected further improvement. This run translated to around 20 seconds (with 10 position fixes per second) which is not long enough for a reasonable trial. The length of the trial was limited by corrupt values found in the data messages from the AllStar unit. The number of satellites detected controls the flow of the program, so a corrupt value here put the program in an effectively endless loop. As the message had a valid checksum, this indicated that the data was corrupt **before** it left the AllStar unit. Marconi was sent a bug report of this problem, but no response was obtained. To get around this problem, the data messages were checked for unlikely values. Corruptions were thus detected; the faulty packets were thrown away and the software continued robustly.

Now that longer runs were possible, it was noticed that it took a long time for the system to find

enough satellites common to both receivers to perform a differential position fix and indeed often this never happened. The problem was traced to the GPS board inside the wearable computer being used. Electrical interference from the other subsystems within the wearable was severely compromising the reception of satellite signals. Eventually when it was realised that even CPU activity was interfering with satellite reception, further software development was abandoned as futile. A request was placed for the wearable to rebuilt in a manner that was not susceptible to interference.

Eight months later the wearable computer returned with additional screening in place. The delay was particularly frustrating as there had been already an eight month wait for the mobile AllStar board to be installed in the wearable computer. On testing, the problem was found to be still present. A further request was placed for the GPS board to be taken out of the wearable — with just RS232, antenna and power connectors so it could operate as a standalone unit. A couple of months later this hardware modification was accomplished and development work could proceed. Both GPS boards were connected to the RS232 ports of a desktop PC instead. At this stage it was noted that the corruption of AllStar messages stopped. The move was from a 486-based to Pentium host, so the explanation of the message corruption problem is that it happens when the rate at which data can be removed from the AllStar unit is not high enough.

For the first time it was possible to undertake longer trials of differential operation. Over the longer timescale, carrier-smoothed differential operation gave very bad results. Initially, a mistake was suspected in the way the carrier phase range was calculated from the integrated carrier phase value provided. Marconi provide a pseudocode description of how to do this which is ambiguous and includes various unexplained fudge factors. Even after considerable study, results were not good. In addition attempts were made, purely on the basis of trial and error, to try and get reasonable figures out of the system. None were obtained. Marconi was contacted for help with the carrier phase calculations, and a request for code (rather than ambiguous pseudocode) was placed. No response was obtained on either account. No Internet GPS contacts had any experience of the unit at this stage.

The eventual suspicion was that the AllStar units were faulty but there was no proof. Instead of giving up, the decision was made to analyse the data coming out of the AllStar units at the **very** lowest level. If sense could be made of these data, perhaps the method of producing valid carrier phase ranges could be inferred *ab initio*? If no sense could be made of the data, then this would be proof of faulty hardware.

Appendix B presents the results of a three month investigation which proved that the data coming

out of the AllStar units was indeed faulty and that one of the units could with good cause be called "broken". The report of this investigation was placed on the Web for ease of dissemination. Marconi was contacted with the findings and directed towards the URL so they could see the visual evidence of the problems. No reply was forthcoming. Marconi's user support line were phoned a number of times. This was never attended, and a number of messages were in consequence placed on their answerphone. The answerphone message promised a prompt response, but no response of any degree of tardiness was forthcoming. At the end of the project it was discovered from an Internet contact in Chile, via an Internet contact in Holland, that the version of the AllStar firmware being used was faulty, and gave incorrect carrier phase readings — precisely as found!

At this stage, the GPS unit which was identified as "broken" stopped working altogether rendering any kind of differential GPS development work impossible. The response to this disheartening circumstance will be addressed in the next section.

4.5 A Fresh Start

Given that only one AllStar board was working — and even that in a somewhat suspect manner, how could a differential GPS system be built? It did not make sense to buy a replacement AllStar board given the total lack of user support, and the flakiness of the AllStar hardware. However, to buy a new candidate differential GPS system would be expensive and necessitate a total rewrite of the device dependent subsystem. Since the purchase of the AllStar board, a later version of Marconi's cheaper SuperStar board had been launched which newly supported carrier phase. Although the SuperStar only supplies readings at 1Hz, instead of the AllStar's 10Hz, the situation was desperate enough that the lower specification system would have been entirely acceptable. The SuperStar board was given serious consideration and indeed requested.

The other possibility was using the "working" AllStar board to deliver a differential GPS correction signal to any common or garden inexpensive handheld GPS unit. Though new software would have to be written, it was a very cheap solution in terms of hardware. Also, as SA had just been switched off, it was time to re-examine the accuracy of a single handheld receiver anyhow. To this end an inexpensive Garmin eTrex handheld GPS unit was purchased and investigated to see how far this technology could take us.

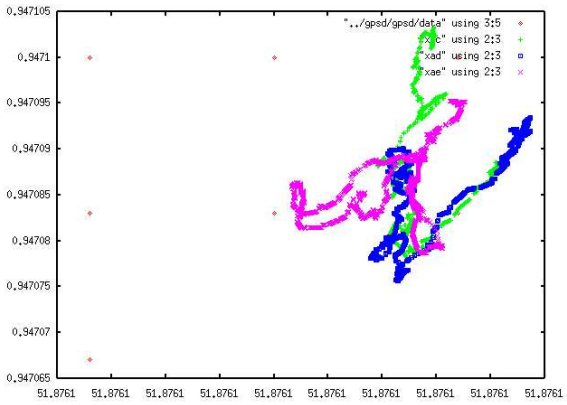

Figure 4.5: Positions reported via Garmin and NMEA protocols (fixed receiver)

4.6 Replacement Equipment

Figure 4.5 shows the distribution of points reported by the eTrex in a stationary position. The graph represents an area of around 5m $\times$ 5m. The horizontal axis is latitude; the vertical axis is longitude. The different types of points represent readings taken during successive time intervals — revealing the wandering nature of the position fix. The communication protocol used to talk to the eTrex was Garmin proprietary — using the MsgPosn message type (id = 17). The points which form a grid are the positions reported through using the NMEA protocol, with the receiver in the same fixed position, over a different long run. The quantisation effect is due to the lower representational precision of this protocol (*i.e.* NMEA *cf.* Garmin).

The representational precision of NMEA is not fixed within the standard: real numbers are represented simply by strings of unspecified length. However, it would seem likely that the number of digits of precision provided in this instance did not anticipate the switching off of SA. Clearly, such absence of precision makes accuracy determination difficult, and does not allow the greater short term relative positional accuracy to be exploited.

For AR a latitude, longitude and altitude are simultaneously required. While the Garmin protocol gives latitude and longitude of sufficient precision, an altitude is not available. While the NMEA protocol provides an altitude, it does not provide a latitude or longitude to the required precision! It is impossible to switch the unit between the two protocols remotely: this change has to be effected on the unit itself.

Garmin user support were contacted regarding this problem, and 1 (and preferably 2) extra digits

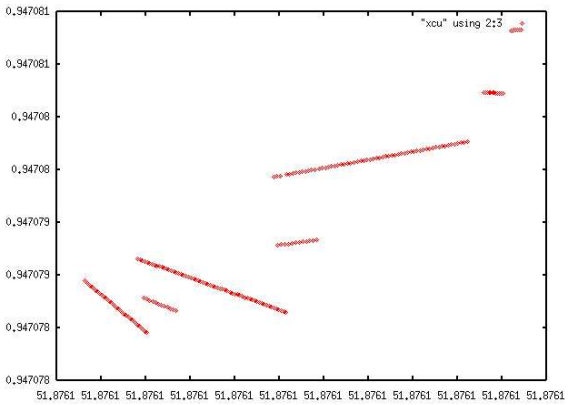

Figure 4.6: Positions reported via Garmin protocol (fixed receiver, short timescale)

of precision were requested in the NMEA protocol. The response was that Garmin would **never** do this as these amounts were below the guaranteed absolute accuracy of the system. One wonders whether Garmin were trying to lock users into their proprietary protocol, because there is no consistency of this argument across protocols. Much later, when the next eTrex software upgrade from Garmin was downloaded, an extra digit of precision had been added to the NMEA protocol! It was established that none of the four RS232 protocols supported by the eTrex can support Augmented Reality applications, despite the fact that the device is physically capable of doing so. Indeed, glaring bugs were found in these protocols and reported back to Garmin.

The eTrex updates its position display every second, and the NMEA protocol sends a position reading every second. However, the Garmin protocol works on a client server basis and the position can be requested multiple times per second. In fact, Figure 4.6 shows a blown-up portion of the graph in Figure 4.5. Horizontal axis is latitude; vertical axis is longitude. The vertical axis represents a distance of around 0.3 metres. Each straight line segment represents a second of data. Instead of this position being fixed for a second, the graph suggests that positions are being extrapolated linearly within the second using a velocity estimate. The irony is that these extrapolations will not be seen by the majority of eTrex users. The standard of this velocity estimate varies, here it is **not** precisely along the locus of the reported position.

Message ID	Message Name	Nature	Software
49	Cmnd_Start_Pvt_Data	COMMAND	DIY
50	Cmnd_Stop_Pvt_Data	COMMAND	DIY
51	Pid_Pvt_Data	DATA	DIY
254	Pid_Protocol_Array	DATA	DIY
27	MsgSatStat	DATA	DIY/public domain
56	Pseudorange Data	DATA	DIY
255	MsgIdData	DATA	DIY
2	ReqPosn	COMMAND	public domain
17	MsgPosn	DATA	public domain

Table 4.10: Garmin position-related message types

4.7 Replacement Device Dependent Software

Given that the eTrex handheld is a much less specialised piece of GPS equipment than the AllStar board from Marconi, it was possible to find public domain source code to talk to the unit. A survey and assessment of all such software was undertaken. For tractability, only those items of software which were found sufficiently useful to be subsequently modified to meet the ends of the project are presented.

1. gd2 (Garmin Protocol)

   `URL: http://vancouver-webpages.com/pub/peter/gd2.tgz`

   "gd2" is unpretentious, text-based 'C' code to download/upload waypoints, and other information from/to a Garmin unit. The "-p" option downloads the required current position. Only a limited number of the Garmin messages obeying the official Garmin protocol are fully decoded. Similarly, only a limited number of the non-official Garmin protocol messages are decoded. Garmin will not release the requisite information for the latter, and the decoding that has been done so far is by hobbyists.

   Table 4.10 shows the Garmin message types that are of relevance to basic position determination: the fourth column indicates whether software support existed in gd2 or had to be added. The Cmnd_Start_Pvt_Data command starts the flow of PVT (Position, Velocity and Time) data in the Pid_Pvt_Data message, while the command Cmnd_Stop_Pvt_Data stops this flow of information. The message type MsgSatStat is only partially supported by public domain code,

and an attempt was made to decode the data further, but fully sensible numbers could still not be extracted from all the data fields. The software modifications carried out were sent to the maintainer of the gd2 program.

2. gpsd + gps (NMEA Protocol)

```
URL: http://www.mayko.com/gpsd.html
```

This software is broken into two parts: a GPS daemon called “gpsd” and a sample client called “gps”. This is an excellent architecture as many different clients anywhere on the Internet can use the same GPS hardware by connecting to the appropriate port on the machine that is physically connected to the satellite kit. For example, you can connect the client to a local RS232 port directly:

```
gps /dev/ttyS0
```

or via the daemon to the remote GPS equipment connected to the local machine “myhost”, say

```
gps myhost.essex.ac.uk 2947
```

where 2947 is the port number. One can also telnet just to observe the raw data coming across (typing “R” initiates the raw data stream) *i.e.*

```
telnet myhost.essex.ac.uk 2947
```

This daemon architecture is a natural for supplying differential GPS correction information over the net — and this program is a suitable starting point. In fact “gpsd” already has some differential GPS support. The sample client called “gps” crashed immediately when first run. This source of the problem was located (a pointer error) and fixed. The GUI code of “gps” was a little suspect because not all the satellite data were being displayed. This is possibly a bug in the underlying GUI toolkit, but the code was modified to remedy this fault.

3. g7 Garmin Protocol (DOWNLOADED + WORKING)

```
URL http://members.home.net/crh24/gps/g7towin/g7towin.htm
```

The is an open source Windows program which reads data from a Garmin GPS unit and writes certain file formats. When difficulties were experienced in detecting PVT messages from the eTrex, an independent test was carried out using the g7 software. Running under a Windows PC, g7 failed to download PVT data from the eTrex. This was the final proof that the eTrex did **not** produce data in this form, despite the fact that a meta-protocol proclaimed that it did. The bug was reported to Garmin.

g7 is the open source code that supports the most Garmin features (but significantly not the secret protocol). However, the source is somewhat sprawling and many system dependencies would make it difficult to port off a Windows platform. Nonetheless, its source was a useful check for decoding many aspects of the Garmin protocol.

4. async + gar2rnx (Garmin Protocol)

   `URL: http://artico.lma.fi.upm.es/numerico/miembros/antonio/pd/`

   These are public domain open source programs available for both for Linux and Windows which work in tandem. “async” records a data stream from a Garmin G12 or G12XL handheld receiver from an R2232 port, to a file with a “.g12” extension. “gar2rnx” translates this file into a standard RINEX file containing pseudorange and phase information. In short the gar2rnx program is the “magic bullet” which decodes the secret Garmin protocols to reveal the information essential for “diygps” calculations. Coincidentally, the program first appeared on the Web during the investigations with the eTrex. The code was naturally tried with the eTrex, but the protocol of the relevant message (id = 0x38) was found to be slightly different, so a new “code cracking” exercise was required. The eTrex was posted to Chris Hill of Nottingham University, who is the UK expert Garmin cracker. Unfortunately, he is not allowed to release full details of his findings because they are built into commercial GPS software called Gringo, which is sold by the University of Nottingham.

   An inexpensive G12XL unit was purchased, and this was proven to work with “async” and “gar2rnx”. The decoding kernel of “gar2rnx” was ported to “gd2”, and the source changes were sent to the maintainer of “gd2”. Similarly the decoding kernel of “gar2rnx” was ported to “g7”, and the source changes were sent to the maintainer. The decoding kernel of “gar2rnx” was also ported to “diygps”, so that this can run using any combination of Garmin G12 and AllStar

units. It proved a major software engineering job to integrate a new device with a different communication architecture into diygps, but this was eventually achieved. In summary, the decoding kernel was taken from the batch processing "gar2rnx" and moved into three real-time codes.

5. dgpsip + dgpsipd (NMEA + RTCM Protocols)

   URL: http://www.wsrcc.com/wolfgang/gps/dgps-ip.html

   dgpsip (standing for Differential GPS Internet Protocol) connects to a port on the Internet that is broadcasting a differential GPS correction signal according to the RTCM protocol. It routes this information down the RS232 port to a connected GPS unit, and fetches back the differentially corrected position from the unit. The architecture is shown in Figure 4.14. The dgpsip program on the local machine partially parses the RTCM messages that come from the correction service, though there is no necessity for the code to do so other than for diagnostic purposes as it only needs to redirect the byte stream to the GPS unit.

   dgpsipd (which stands for Differential GPS Internet Protocol Daemon) is the open source UNIX server software which works with the dgpsip client. dgpsipd multicasts RTCM correction data over the Internet, and collects statistics on where these data are being used. The URL is:

   URL: http://www.wsrcc.com/wolfgang/gps/dgps-ip.html#server

   Differential connection services over the Internet are very skeletal. There are really only three US-based feeds as shown in Table 4.11. The public domain client/server software combination of dgpsip/dgpsipd effectively supplies differential corrections over the Internet, but the suspicion is that the relative expense of base stations has stopped the proliferation of the service.

4.8 Replacement Differential Architecture

Using the combination of the parsing software within dgpsip and the RTCM protocol standard document [47] there is enough information to generate one's own RTCM messages from raw satellite information. How does one test that one has correctly generated RTCM messages? One approach is

host	port	location	equipment
dgps.wsrcc.com	2101	Pt. Blunt, CA USA 37.19 - 122.39	Garmin GBR-21 w. 4ft. E-Field whip
dgps.wsrcc.com	2103	Texas A&M, TX USA 30.60 -96.36	Trimble 4000 (survey grade)
dgps.wsrcc.com	2104	Youngstown, NY USA 43.23 -78.97	CSI MBX-2 w. H-Field antenna

Table 4.11: Differential correction feeds on Internet

to use the information parsed by dgpsip to regenerate the RTCM message and then check for byte-for-byte reproducibility of the input and output streams (Figure 4.7). To ensure a separation of concerns, well-defined APIs are positioned between the parsing and generating software components: these are addressed later.

Closely monitoring the functioning of dgpsip shows the byte stream actually consists of DATA bytes forming RTCM messages and CONTROL bytes which form Garmin proprietary control messages. The eTrex will not switch into differential mode without these proprietary control messages. So much for having a standard like RTCM. This is why presumably, the eTrex will **only** recognise the California Internet beacon (and not the Texas or New York ones). Only the California Internet beacon is a Garmin one. Note that unless the live data streams had been monitored off the Internet, it would have proved impossible to provide RTCM correction to Garmin hardware *i.e.* there is not sufficient documentation.

The DATA and CONTROL messages are arbitrarily mixed on a byte-by-byte basis, so after parsing it is impossible to regenerate this mix. Is it necessary for successful operation to re-intermix the messages in any particular way? Fortunately, this was found experimentally not to be the case. The next best approach is to go for byte-by-byte reproducibility of the DATA and CONTROL byte streams individually. This is easily achievable for the CONTROL bytes but not for the DATA bytes. The NMEA protocol consists of messages of 30 bit words. The bottom 6 bits of each DATA byte are data. One would expect each 30 bit word to be formed from 5 DATA bytes. Wrong! The DATA stream is arbitrarily shifted at the bit level, so individual DATA bytes may hold the end of one message, and

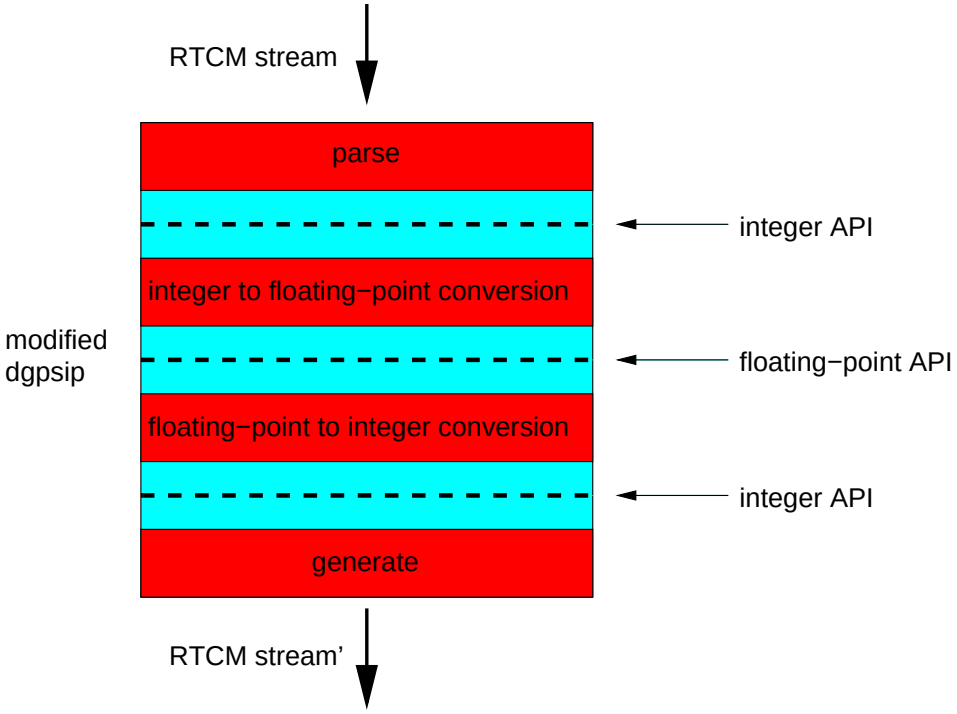

Figure 4.7: Debugging external APIs using byte reproducibility of RTCM stream

the beginning of another message. The degree of the shift is unknown in advance, so byte-by-byte reproducibility had to be investigated by testing against all of the 6 possible bit-rotation options!

For *ab initio* message generation, this shifting could be switched off altogether to simplify operation. Would shifting an existing message still yield successful operation? Fortunately, this was found to be the case. Of course, for comparison purposes during debugging the ability to replicate this shifting is necessary. The code was written to automate the testing of byte-by-byte reproducibility of the CONTROL and DATA streams. After much software effort this test was finally passed successfully. The payback justified the effort: the resulting information stream was immediately and successfully interpreted by the eTrex as a differential correction message, despite the separation of the DATA and CONTROL bytes. Interestingly enough, once this work was completed, it was noticed that the RTCM data stream coming from California over the Internet has switched off the bit rotation.

The software was then ready to be modified to work with satellite data in the form they come from the AllStar. In preparation, further small conversion layers were produced (Figure 4.7) which convert to and from a floating-point representation respectively. Although, the RTCM protocol is purely

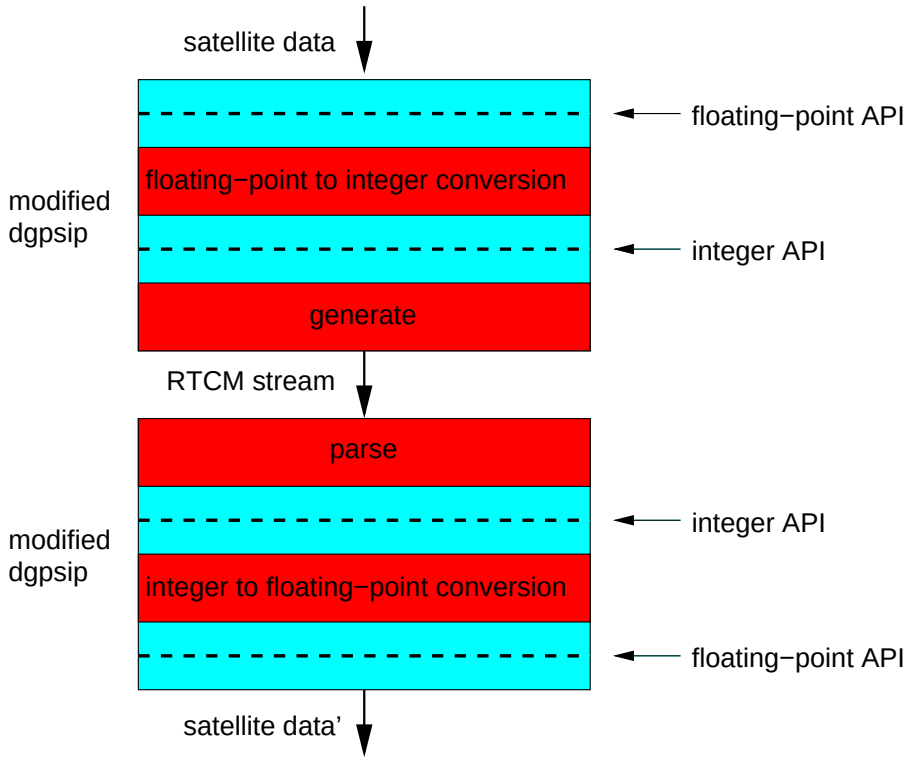

Figure 4.8: Debugging external APIs using reproducibility of satellite data

integer it is necessary to go through a preparatory floating point interface, as the AllStar provides floating-point data. The input was checked against the output to verify a null transformation.

The complete system was also debugged by setting up the RTCM generating software on the Internet as a byte stream, and checking if another copy of dgpsip can correctly parse this (Figure 4.8). This time the input and output satellite data are compared to check that the null transform has been effected. Note that arbitrary satellite data will not be regenerated exactly because they go through an intermediate integer representation in the RTCM data stream so accuracy will be lost. Once it was shown that the satellite data were approximately regenerated then then the eTrex could be put in its rightful target position (see Figure 4.9).

The eTrex reported that it was indeed performing differential position calculations using correction data derived from the AllStar board. However, the position fix did not seem to be improved or the EPE (Estimated Positional Error) as reported by the eTrex did not seem to be particularly reduced, despite the fact the error in the pseudorange correction (part of the RTCM protocol) was explicitly set to zero.

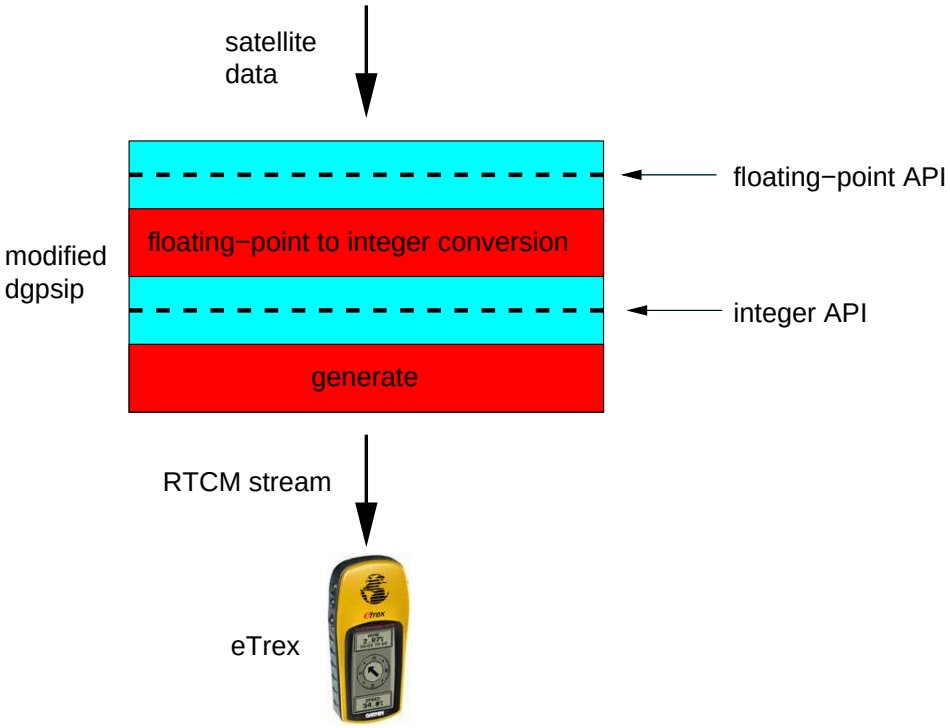

Figure 4.9: Target software configuration

A typical pseudorange correction from California was 5 metres; a typical pseudorange correction from the AllStar was around 2000 metres. It was realised that pseudorange correction from California was calculated only after a few standard corrections have **already** been applied to the pseudorange. Part of the reasoning behind this must surely be to keep the actual number transmitted small. There are different conventions as to what constitutes a “pseudorange difference”, and no indication could be found as to which corrections had been applied already. If the correction terms used in the standalone position calculation are applied, then the DIY differential correction is reduced to around the size of those coming from California. However, there is no way to verify correction terms produced in Colchester against those sourced from California!

It was realised that initially one should build a complete DIY differential GPS system rather than simply a component of another system. Only once the complete DIY system is working, is it advisable to integrate parts into other systems — especially if these other systems have unknown conventions.

It is noteworthy to report that on a single occasion, a differential correction produced in California was usable in Colchester, because at least 4 satellites in common could be viewed from both locations.

Naturally, no effective increase in positional accuracy was obtained due to the fact that the baseline was (by far) too long.

dgpsip also routes the differentially corrected position fix from the GPS unit back to the dgpsip server (in California or wherever). This information is used to see who is using the dgpsip service, but also as a way of knowing whether the server is connected to a real-time system *i.e.* one that can respond with a position fix. If a responding system is detected then the dgpsip server works in real time and does not buffer up the information stream. If one connects to the dgpsip server using telnet, say, the screen output coming from these servers will appear batched for this reason. Clearly the dgpsip server is trying to maintain responsiveness for the real-time users throughout the globe.

dgpsipd was modified so that instead of taking corrections from a differential GPS base station it computes corrections taking raw pseudorange data from an AllStar board or a Garmin G12(XL) handheld unit. The is achieved very simply by piping diygps and dgpsipd together *i.e.* running "diygps | dgpsipd". A small parser was written in dgpsipd to pick up the appropriate output of diygps.

The capability of the differential GPS software that has been built undermines some of Marconi's products. For example, the version of the AllStar board which provides RTCM correction messages is much more expensive than the basic model that was purchased. This software which turns the cheaper product into the more expensive one, has been placed on the Internet. Now we have produced a Internet-based differential correction service using inexpensive GPS equipment. Only the source for a single client version of dgpsipd is available on the Internet, but it would be a relatively simple exercise to turn this into a multiple client program. However, a server capable of serving only one client at a time is sufficient to check out the working of our client-server application.

4.9 Original Differential Architecture Revisited

The original differential architecture envisaged was revisited through the purchase of an second inexpensive Garmin G12XL receiver, given the confidence established in the data emerging from this unit. The resulting and final DIY differential GPS system developed can use any combination of AllStar boards (if any non-faulty ones exist) and G12 units, although one AllStar board must be present to provide ephemeris data, which the G12 does not provide. The software has been correspondingly extended to support three abstract devices: a reference GPS device; a mobile GPS device and an ephemeris device. If neither the reference nor the mobile device supplies ephemeris, then this can be

Per Satellite Data					
Symbol	Garmin I/O S/W Structure Element	Computational Kernel Variable (arranged as batch file input)	Garmin I/O S/W Structure Type	Garmin Message Type	Location in Satellite Data
P_{rs}	`pr`	`SV[sv] or P[sv] = pr;`	type_rec0x38	56	N/A
C_r	`int_phase and c_phase`	`C[sv] = ( int_phase +(c_phase & 2047)/2048.0 ) * 0.190293672;`	type_rec0x38	56	N/A
Per Position Calculation Datum					
T_{rc}	`tow`	`Trc`	type_rec0x38	56	N/A

Table 4.12: Link between Garmin satellite hardware and software

taken from a dedicated ephemeris device.

In practice, it was decided to use pseudorange and carrier phase ranges from two Garmin G12XL units to circumvent any problems with the quality of the AllStar data. In retrospect, this proved a wise move, given the subsequent discovery of the faulty AllStar firmware. The Garmin units supply P_{rs}, T_{rc} and C_r for each satellite in view, while the AllStar unit supplies all other parameters to the GPS calculations. This solution conveniently avoids a total revamp of the interface code, but in actual fact the other parameters necessary for the calculation are simply **not** obtainable from the Garmin unit. The secret Garmin protocols may be further cracked in future to yield this data — allowing a DIY differential GPS system to be built purely from two Garmin G12XL units. This situation would be exceptionally desirable. The summarised details of interfacing Garmin hardware into the GPS computational kernel are given in Table 4.12. A basic PC only has 2 RS232 ports, so how is it possible to connect to a Garmin G12XL reference device; a Garmin G12XL mobile device and an AllStar ephemeris device? The solution is to connect the supernumerary devices to other machines on the network and to connect to these devices via sockets as shown in Figure 4.16.

Astoundingly, no software was found on the Web to turn a readable/writable UNIX device into a socket on the network, so such a program called `dev2soc` was written. `dev2soc` permits serial access by a number of different clients to the networked socket. Bi-directional buffered transfer is effected, which removes many of the problems causes by applications failing to read data from the RS232 ports in time, because they are elsewhere doing computation.

For every client, `dev2soc` forks off a process called “do_something” which splits into two copies

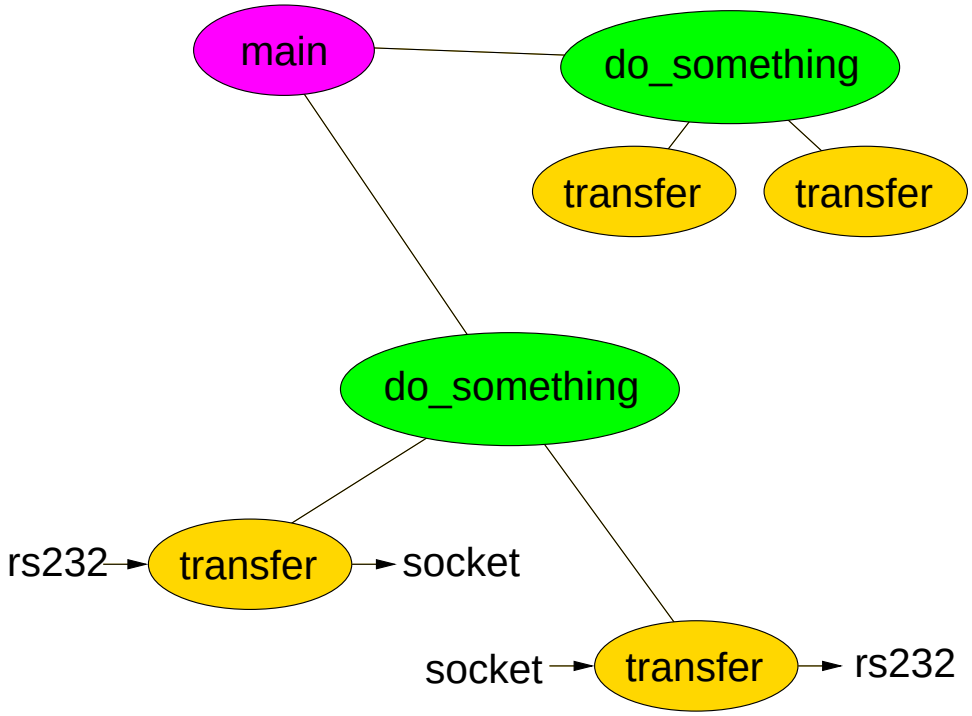

Figure 4.10: Process structure of `dev2soc`

of a process called “transfer”, each of which deals with communication in a single direction. The process architecture is shown in Figure 4.10. The code for “transfer” is presented in Appendix D. Multi-client writing will create the expected clash of resources, but multi-client reading is fine. The applicability of `dev2soc` goes far wider than the GPS application, and the code has been mounted on the Internet. Documentation for the utility is given in Appendix D.

4.10 Real-Time Architecture

The real-time design issues of a GPS system are more complex than they at first appear. For Type 1 receivers (such as the AllStar) temporally corresponding pseudoranges from two units are easily determined by the timestamps being close together. This also holds true for Type 2 boards, where clock errors are only of the order of a few milliseconds. Few GPS systems take measurements at kHz frequencies. However, with Type 3 receivers (such as the Garmin G12), the timestamp drifts away from true GPS time forever. Establishing whether times from 2 different units correspond is

not simple — they may be several seconds adrift or more. This has been witnessed during longer term trials. The diygps program deals with inputs from a number of devices. At any instant, how does it decide from which device to read next? The synchronisation requirements for an RTCM-based differential correction architecture are not as stringent, by design, as for the DIY system, however, they still have to be addressed. Summarising these real-time requirements:

1. the data from the different devices must be synchronised for computation to be valid
2. all devices must be serviced so data is not lost.

4.10.1 Type 1 Receivers

If the same control instructions are sent to both receivers, then the same number of messages will come back from each. However, even if the receivers are serviced in a round-robin basis, synchronisation may be lost due to a dropped message. A more sophisticated approach looks at message time stamps. The function order_timer returns true if its time arguments are in order or are approximately equal, within the limits of the supplied threshold argument *e.g.*

$$order\_timer(Trc_{rs}, Trc_{ms}, 0.05) \Leftrightarrow Trc_{rs} < Trc_{ms} \text{ or } |Trc_{rs} - Trc_{ms}| < 0.05$$

where Trc_{rs} is the time stamp of the last reference receiver pseudorange message; and Trc_{ms} is the time stamp of the last mobile receiver pseudorange message. If the pseudoranges from the reference receiver are older than the pseudoranges from the mobile receiver, then it is correct that the reference receiver should be read next. If the last pseudoranges read are of the same vintage, then either port can be read — the reference receiver is chosen in the code below:

```
while ( 1 )
{
   if ( order_timer(Trc_rs,Trc_ms,0.05) ) { /* in order or equal */
     // read from reference receiver and put new timestamp in Trc_rs
   }
   else {
     // read from mobile receiver and put new timestamp in Trc_ms
   }
}
```

This is fine for Type 1 receivers whose clocks are mutually synchronised, through being individually synchronised with GPS time. The fastest rate at which pseudorange data is received from the

AllStar is 10 Hz, so a suitable threshold is half this period *i.e.* 0.05 seconds.

4.10.2 Type 3 Receivers

When Type 3 units are being used, the situation becomes much more complex as their clocks can drift infinitely, and hence infinitely apart. The timestamps chosen are thus the corrected values of Trc_{rs} and Trc_{ms} derived from the standalone GPS position calculations. If there are insufficient satellites (*i.e.* < 4) to perform a position fix, then the synchronisation of messages may go temporarily wrong. However, in this case there is no possibility of a differential position calculation anyhow which is after all what the system is all about. Clearly, the code above must be generalised to look for the lowest corrected timestamp from 3 GPS receivers when there are separate reference, mobile and ephemeris devices. Similarly to trigger a differential calculation, the condition

$$|T'rc_{rs} - T'rc_{ms}| < \frac{1}{2f}$$

must be true where f is the frequency of the pseudorange readings and $T'rc_{rs}$ and $T'rc_{ms}$ are the corrected reception times of the last pseudorange messages for the reference and mobile receivers respectively.

4.10.3 Replacement Differential Architecture

The ephemeris data and RTCM data must also be of the appropriate corresponding vintage for the replacement differential system developed. Ephemeris data are updated only relatively infrequently (it contains predictive orbital parameters) and are incorporated into calculations as soon as they are transmitted by the satellites. Each block of ephemeris data is given an IOD or Issue of Data number, which increments by one each time.

RTCM corrections must be delivered within a certain timescale to retain validity. Broadcast methods have traditionally been slow *e.g.* low bit rate maritime beacons. To this end RTCM also contain predictive parameters to calculate the time varying pseudorange corrections (though this is no more than a simple linear model). Also, each RTCM packet contains the corresponding ephemeris data IOD, so very old (and hence invalid) corrections can be rejected.

This is why totally fabricated RTCM data **cannot** be sent to GPS units to test the protocol, because the IOD needs to be valid. In our replacement DIY system running on the Internet, the RTCM channel

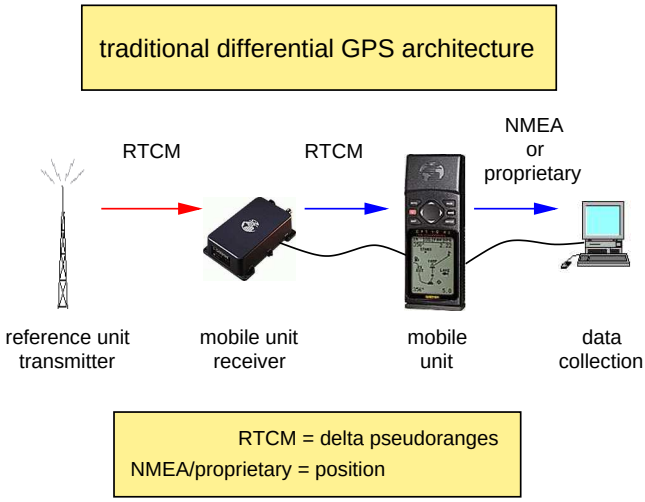

Figure 4.11: Traditional architecture

did not suffer from bandwidth problems as the frequency of RTCM messages was high — removing the necessity for predictive terms. Late delivery of ephemeris data is not going to occur, and the use of slightly untimely ephemeris parameters will not cause a problem. Late delivery of RTCM data is also not going to occur, but very late delivery would be safely rejected anyhow.

4.11 GPS Architectures

It is worthwhile stepping back from the particular differential GPS systems developed, to look at the architecture of such systems in general to gain a more abstract insight in order to propose improved systems.

4.11.1 Traditional Architecture

The traditional differential GPS architecture is shown in Figure 4.11. This reveals the largely maritime origins of differential correction services, where maritime beacons broadcast the correction signals in the RTCM protocol. It is essentially asymmetric, with totally different reference and mobile receiver data paths. The RTCM protocol is designed for low bandwidth technologies, and only allows a stylised differential correction which ultimately limits achievable accuracy. There is an “unnecessary” wireless

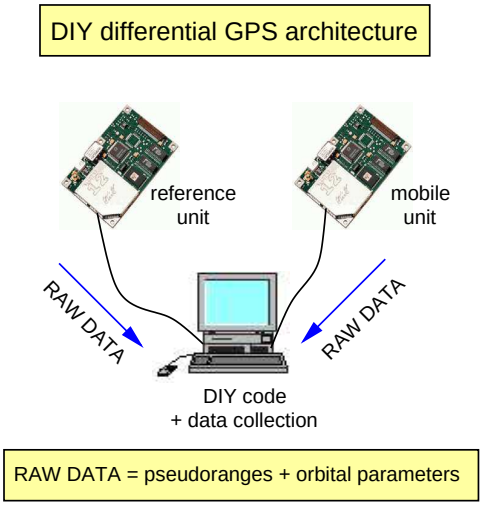

Figure 4.12: DIY architecture

stage - obligating the purchase of an additional (generally terrestrial) radio receiver. Various systems are used to transmit the corrections: FM wireless, satellite, maritime beacons and even amateur radio!

4.11.2 DIY Architecture

Figure 4.12 shows our original DIY architecture. The ultimate intention is for the mobile receiver to be connected to a wearable computer, and the final system would involve a wireless link to a PC base station. With a generic wireless computer network, the positioning of software components and software interfaces is arbitrary. For example, the computation could be carried out on the wearable computer or on the computer at the base station. However, the diagram shows the essential connectivity pattern, where the rôles of the reference and mobile receivers are essentially symmetric. Indeed the divisions of rôles, is based purely on how the units are used not on the architecture of the system. This architecture is more satisfactory for accuracy, as the complete raw data are globally available.

4.11.3 GPS Position Server

Figure 4.13 shows a non-differential GPS architecture. However, the server-based approach for a single receiver, shows the way ahead for differential services. Instead of talking to the hardware directly in a proprietary protocol or via NMEA, the client talks to a server via the simpler GPSD protocol.

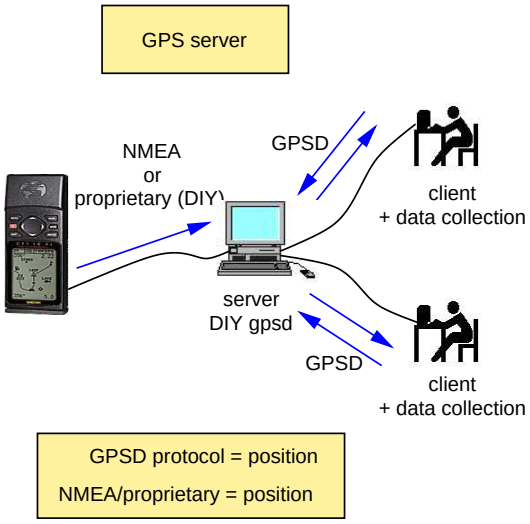

Figure 4.13: GPS position server

Many clients can connect to the server simultaneously, and an instantaneous result is available (*cf.* waiting for the hardware to respond or for the next position fix — typically one a second). For the NMEA protocol, the *de facto* approach is to use the gpsd daemon to connect to the hardware, so the use of gpsd and an appropriate client is recommended. dgpsip support has also been added to gpsd.

4.11.4 Internet Correction Service

Figure 4.14 is similar to the traditional architecture except that the radio broadcast is further transformed into an Internet multicast. It is necessary to connect a computer to the correction receiver, so corrections can be routed out on the Internet. Another computer accepts these corrections and passes them on to the mobile receiver. Note that client positions are sent back to the correction server and it is worthwhile emphasising that this “feedback” route could prove much more useful in the future. This capability for feedback is the novel feature of differential correction over the Internet, The client and server software (dgpsipd and dgpsip respectively) are public domain programs.

4.11.5 DIY Internet Correction Service (Our Replacement Architecture)

The computer connected to the reference receiver, generates and sends the corrections out on the Internet (Figure 4.15). This is similar to the usual Internet connection service but there is no wireless

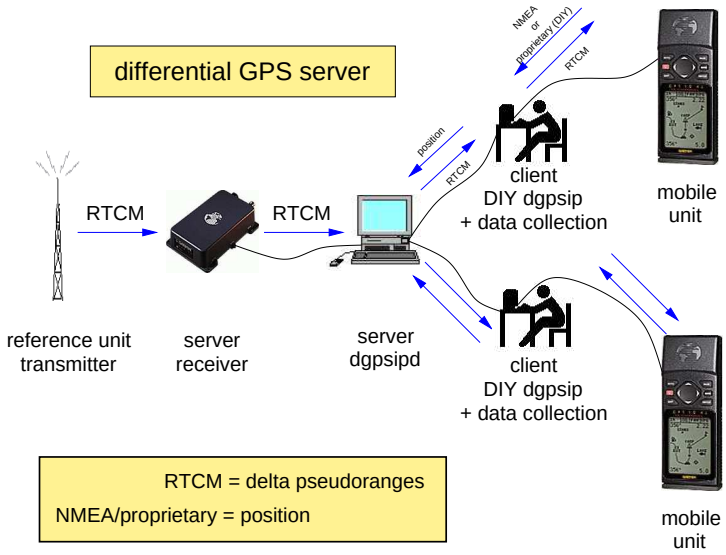

Figure 4.14: Internet correction service

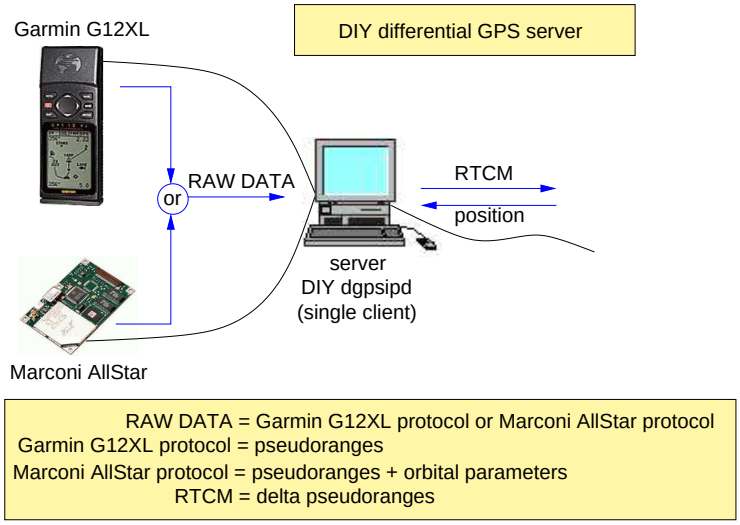

Figure 4.15: DIY Internet correction service

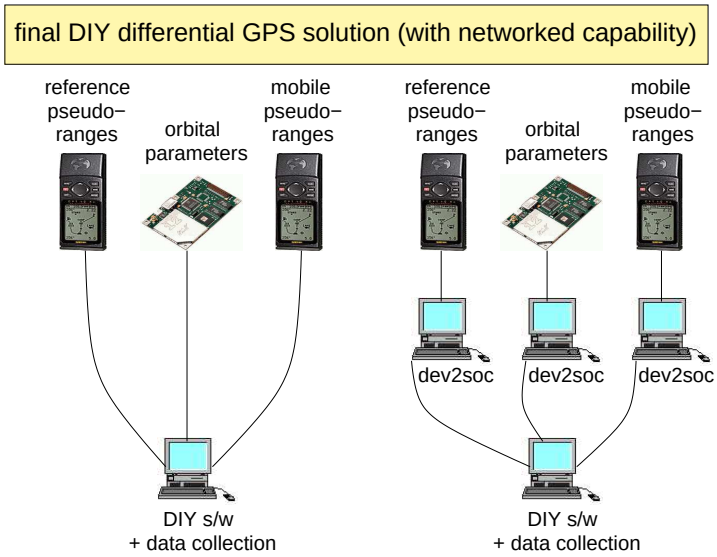

Figure 4.16: Final DIY architecture

stage, removing latency. The public domain program dgpsipd was modified to produce a correction signal from cheap AllStar and Garmin G12XL units. The existing version of dgpsipd has to be used with more expensive dedicated base units which generate the appropriate RTCM protocol directly. Note that for convenience, much of the development work was done with dgpsip, and then the verified software was moved over to the target dgpsipd executable.

4.11.6 Final DIY Architecture (Original DIY Architecture Revisited)

Figure 4.16 shows the final DIY GPS system developed. Due to faulty hardware, the new architecture is obliged to draw information from three devices. One device provides orbital parameters for the satellites (ephemeris data) and the other two devices provide pseudorange information. The diagram on the left shows the virtual connections. The diagram on the right indicates that the devices themselves may be plugged into any computer within the network. A small specially written utility called `dev2soc` connects a remote device to a socket on the remote machine. The position calculation software can either connect to a device directly on an RS232 port or indirectly via a socket.

As typical PCs only have two RS232 ports, one of the three devices must be placed remotely. While the separation of concerns (*i.e.* having both pseudorange devices and an ephemeris device) is administratively inconveniencing, there are attendant advantages. For example, a GPS unit that only

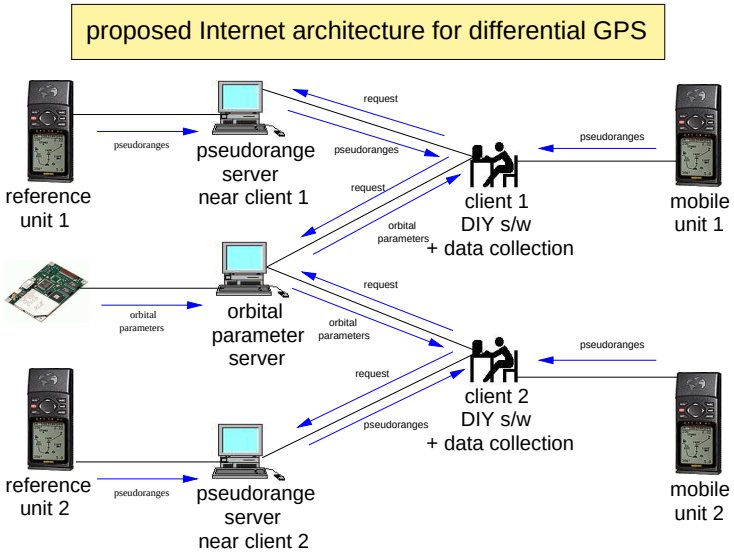

Figure 4.17: Proposed Internet architecture

produces pseudoranges is much simpler, cheaper and can cope with poorer reception conditions, than one which extracts ephemeris data from the signal. This is the principle behind the so-called “server-assisted differential GPS”: the mobile unit can work in locations of low signal strength (sometimes even indoors) while the calculation is shipped off by wireless to a server which has access to full ephemeris and other information.

4.11.7 Proposed Internet Architecture

This architecture takes the final DIY architecture a stage further, by automating differential GPS correction for all Internet users. The proposal takes the form of a number of components shown in Figure 4.17:

1. Orbital Parameter Server

   The orbital parameters are global information. They do not vary locally, like pseudoranges, so a single orbital parameter server would suffice for the planet. A network of only 4 well-sited receivers (arranged tetrahedrally) would cover the Earth. It is almost inconceivable, from the benefits it would confer, that this information is not made available live on the Internet! In fact, the ephemeris data are uploaded to the satellites from ground stations monitoring the orbits, so this data could be made directly available on the Internet by the American DoD.

2. Pseudorange Server

   In order to perform accurate DIY differential GPS calculations it is necessary to connect to a local pseudorange server, that is within say 20 miles. Either:

   (a) an appropriately dense network of these could be situated throughout the globe or

   (b) a less dense network could be situated throughout the globe (using interpolation to estimate a local pseudorange) or

   (c) a sufficiently close base station could be mounted on the Internet, as and when required.

   Our final DIY differential GPS system, is essentially (c), where a reference station would be installed on the Internet, located at Gosbecks Archæological Park.

   Ideally, one would connect to a single virtual pseudorange server, and the actual connection would be made transparently to a particular local server. This is a case where feedback information from the client would be vital to control this transparent indirection. A single receiver position fix from the client's unit would identify its location to the virtual server, which could then assign the appropriate pseudorange server.

   An alternative computational method could be based on the same architecture. All pseudoranges could be sucked back from all GPS units connected to the Internet. Instead of merely having a reference and mobile receiver there is now access to a full network of receivers spread around the globe. Finding the most likely position fix for all the receivers and indeed the satellites is literally a huge global optimisation process that could yield exceptionally high accuracy. It may help if those fixed receivers (at surveyed positions) are appropriately tagged. "Fixed" receivers could simply be tagged with a "time since last moved", and the global optimisation code would know that its average position since this move (if sufficiently long ago) could act as surveyed position. Indeed, if fitted with an accelerometer, say, it would be possible to automate finding out how long a GPS unit had been in a fixed position. Naturally, to trap the injection of rogue information that might scupper a global system, it may prove necessary to introduce encryption.

4.12 Assessing Accuracy

GPS code inherently involves a lot of corrections or "tweaks" to try to improve the accuracy. It is necessary to assess these corrections in terms of the resulting quality of output. What is the best way

to determine whether an intended correction is for the better or the worse?

The scatter plot of positions reported for a fixed receiver was the first approach. This gives an indication of quality of a GPS system but is not a simple or instant metric. The "average distance from the average position" has also been used which is a single figure describing the tightness of clustering. However, it has proved impossible to achieve a better average accuracy than 1.8 metres (over typical timescales for interaction) using this metric with the differential GPS system that has been built. Is this a limitation of the hardware or the software? To answer this question requires a different approach, namely analysis of the quality of the low-level data emerging from the hardware.

The hardware essentially supplies 3 numbers for each satellite in view:-

1. t: a measurement time
2. P: a pseudorange measurement giving the time of flight from each satellite to the GPS receiver as a distance
3. C: a carrier phase measurement giving a more accurate (but relative) pseudorange measurement

The position of a satellite is essentially calculated from only t alone (an orbit is deterministic). Knowing the position of the receiver, then the actual distance to the satellite $R(t)$ can be calculated as the distance between these two points. The correction factor in a differential GPS system is $(R(t)-P)$ which exactly corrects the measured pseudorange to be the actual distance. This correction can also be used to improve the pseudorange measurement of the same satellite from a second nearby receiver, and this is the basic idea behind differential GPS.

For the sake of the current argument C will be ignored, though this is used in practice to enhance the accuracy of P to give a smoothed pseudorange S. One would expect the term $R(t) - P$ to be relatively steady with time, but measurements as in Table 4.13 show a larger variation that expected from one second to another. The variations are of the order of the pseudorange corrections that are encountered in a typical RTCM message, which are valid for a reasonable time interval. The message would appear to be that basic pseudorange measurements from the Garmin G12XL are not a particularly good source of differential corrections. What is responsible for this noise: is it the inaccuracy in the measurement of t; the measurement of P; or in the code which calculates R ?

Actually, the situation is not as clear-cut as this, because the clock on the Garmin 12XL is known to run "fast" so t will be **wrong** and this will affect the value of P which will be stupidly large. So before calculation of $R(t) - P$ begins, t and P will be corrected by the calculated clock error (which

Correction (metres)	Time (seconds)
4.15	226020.927292
4.34	226021.927292
0.71	226022.927292
1.62	226023.927293
1.79	226024.927295
1.45	226025.927295
1.18	226026.927296
0.39	226027.927296
-0.72	226028.927296
-0.98	226029.927298

Table 4.13: Corrections calculated from basic Garmin pseudoranges

is worked out in a global least squares calculation looking at the value of P from all satellites). So we have to change the question to: does the inaccuracy lie in the measurement of t; the measurement of P; the code which calculates R or in the clock correction code? Let us call the value of the clock correction dt. Let t' and P' represent the corrected values of t and P. To work out where the "wobble" in $R(t') - P'$ comes from we can look at each term independently, with and without clock correction. This gives us 4 terms:

1. $R(t)$ depends on code to compute R and t
2. $R(t')$ depends on code to compute R, t and dt
3. P depends on P
4. P' depends on P, t and dt

How do we assess the wobble in each of these terms, when each is changing rapidly? A satellite moves rather quickly! A method of differencing was devised for the task. Take the n^{th} order difference until the $(n+1)^{th}$ order difference is larger or equal in magnitude to its predecessor. n is then an indication of the noise in the data, if it is expected (like a pseudorange) to vary smoothly. Averaging the values of n over a series of readings will provide an even better measure. To illustrate, using satellite number 2 and time correction for R and P, the differences in Table 4.14 were produced. For this particular set of differences, the quality of $R(t')$ is 6 and quality of P' is 2.

Table 4.15 shows the quality metric for R, P, C and S and their time corrected equivalents, where the clock correction was worked out from the set of P from visible satellites. For the sake of argument

Difference Order	$R(t')$	P'
0	21702995.5523380004	21784545.7962509990
1	-218.0786719993	-218.2540440001
2	0.0991910025	0.8254990019
3	0.0004290044	5.2533770055
4	0.0004180036	13.5920390114
5	0.0004099980	25.4696670212
6	-0.0000300109	39.0663280375
7	-0.0000300109	39.0663280375
Quality (n)	6	2

Table 4.14: Quality metric derived from differences of $R(t')$ and P' from Garmin receiver

Quality Metrics - Clock Corrections Based on P						
n-th order difference quality table (over 1000 readings)						
Variable	Satellite no					
	2	9	10	17	23	26
$R(t)$	3.582	3.516	3.630	3.445	3.529	3.603
$R(t')$	3.514	3.528	3.651	3.408	3.526	3.584
P	2.360	2.314	2.306	2.353	2.305	2.370
P'	2.326	2.284	2.288	2.307	2.302	2.292
C	3.215	2.892	3.191	3.132	3.006	3.230
C'	2.306	2.287	2.287	2.303	2.291	2.288
S	3.211	2.874	3.167	3.145	3.021	3.210
S'	2.724	2.521	2.672	2.793	2.603	2.722

Table 4.15: Quality metrics (Garmin data, clock correction based on P)

examine satellite number 2. The quality of the raw carrier phase data (3.215) is much higher than the quality of the raw pseudorange data (2.360) but not as high as the "correct" (or calculated) distance (3.582). This is as expected, and supports the validity of the metric. It is interesting to speculate why the correct distance is not of infinite quality. Possibilities include:

1. Error in time measurement used to calculate distance
2. Finite numeric precision
3. A measured time varying real-world quantity is inherently not of infinite smoothness

The first is probably the largest component. This could be investigated but the greater concern is with achieving relative quality increases. Correcting for clock error in the GPS receiver, the calculated

Quality Metrics - Clock Corrections Based on S						
n-th order difference quality table (over 1000 readings)						
Variable	Satellite no					
	2	9	10	17	23	26
$R(t)$	3.576	3.507	3.625	3.444	3.521	3.597
$R(t')$	3.463	3.441	3.625	3.388	3.479	3.549
P	2.359	2.315	2.310	2.353	2.306	2.374
P'	2.298	2.301	2.309	2.338	2.309	2.322
C	3.215	2.892	3.191	3.132	3.006	3.230
C'	2.734	2.525	2.681	2.798	2.617	2.738
S	3.211	2.874	3.167	3.145	3.021	3.210
S'	2.724	2.521	2.672	2.793	2.603	2.722

Table 4.16: Quality metrics (Garmin data, clock correction based on S)

distance ($R(t) \rightarrow R(t')$) does not particularly lose quality ($3.582 \rightarrow 3.514$). Correcting for clock error in the GPS receiver, the pseudorange ($P \rightarrow P'$) does not particularly lose quality ($2.360 \rightarrow 2.326$). In conclusion, both the major error in $R(t) - P$ is P and the major error in $R(t') - P'$ is P'. As P' appears to be derived from P (a measured quality) without appreciable loss in quality, the conclusion is that the limiting factor in the differential GPS calculations using pseudorange data alone **is** the quality of the pseudorange data itself rather than the time measurement or any problem with the software. Basically, the hardware is the limiting factor.

On the other hand, correcting for clock error in the GPS receiver, the carrier phase ($C \rightarrow C'$) **does** lose quality ($3.215 \rightarrow 2.306$). The metric tells us this is the **wrong** thing to do. In fact this makes sense (with hindsight of course) because the clock correction is derived from the pseudorange (a cruder measure). Correcting for the clock error in S does loses quality but not so much, presumably as S is an averaged quantity.

Table 4.16 shows the quality metric for R, P, C and S and their time corrected equivalents, where the clock correction is this time worked out from the set of S from the visible satellites. Discussions are again based on satellite 2. The results show that clock correction using S, does not degrade the quality of R appreciably ($3.576 \rightarrow 3.463$) and does not degrade the quality of P appreciably ($2.359 \rightarrow 2.298$). The quality of C is degraded somewhat ($3.215 \rightarrow 2.734$) but not as much as before (*cf.* $3.215 \rightarrow 2.306$). This time $Q(C') > Q(P')$ whereas before $Q(C') \approx Q(P')$ where $Q(x)$ is the quality of quantity x.

The quality of S mirrors that of C showing that they hold the same level of quality — slightly more before clock correction and slightly less after clock correction. Indeed, we want S to have the same quality as C (which is of higher quality than P) as this is the whole point of data smoothing! The result suggests using S for clock correction instead of P, so the benefits of using the phase information C will then not be totally lost.

4.13 Refinements in Calculation

4.13.1 Which Clock Correction?

The differential GPS position calculation was performed both with and without carrier phase smoothing; with clock correction based on P; with clock correction based on S and without clock correction. When a clock correction is made ($T_{rc} \rightarrow T'_{rc}$) corresponding range corrections $P' = P + (T'_{rc} - T_{rc}) \times c$ and $C' = C + (T'_{rc} - T_{rc}) \times c$ are also required. The effect on positional accuracy of these changes is difficult to assess. In contrast the work on low level methods above where using the quality metric makes it very clear whether particular corrections are beneficial or not. For each of the two types of time correction, the positional calculations were carried out in four different passes as summarised in the following table.

pass	differential calculation	
	pseudorange used	reception time used
0	P	T_{rc}
1	P'	T'_{rc}
2	$S = f(P, C)$	T_{rc}
3	$S = f(P', C')$	T'_{rc}

4.13.2 Can Clock Corrections be Avoided?

If inaccurate clock corrections cause problems, is it possible to avoid any? The calculation for a satellite's position goes like this:

$$\tau = \frac{P}{c}$$

where τ is the flight time of signal; P is the pseudorange; and c is the speed of light.

$$T_{tr} = T_{rc} - \tau$$

where T_{rc} is the time of signal reception (measured); and T_{tr} is the time of signal transmission. The position of the satellite is given by T_{tr} (a deterministic orbit is purely a function of time). Consider an error in the receiver's clock of dt. The pseudorange will be too big by $c \times dt$. Let the primed quantities represent the time corrected quantities with error.

$$\begin{aligned} T'_{tr} &= T'_{rc} - \tau' \\ &= T'_{rc} - \frac{P'}{c} \\ &= (T_{rc} + dt) - \frac{(P + dt \times c)}{c} \\ &= T_{rc} - \frac{P}{c} + (dt - dt) \\ &= T_{rc} - \tau \\ &= T_{tr} \end{aligned}$$

Basically, any clock error will factor out in the calculation of T_{tr} and hence in the calculation of satellite position. Well, not quite! In the time the signal takes to travel, the earth (in which the coordinate system is fixed) will have rotated a little. So the satellite position, will have to be rotated round the earth's axis by $\tau \times \Omega$, where Ω is the rate of rotation. There is a small dependency on τ which prevents the clock error from factoring out completely. The most accurate strategy in determining satellite position, is to use $T'_{tr} = T'_{rc} - \frac{P'}{c}$ for T_{tr} (as clock errors factor out here). When we do have to use a quantity that is affected by clock error *e.g.* τ only then do we apply clock correction, and clock correction based on S rather than P, which has been shown to give the better quality result.

In summary, clock correction has only to be applied explicitly in one part of the differential calculation: that of working out the time of flight of the signal τ. Although it is possible to apply clock correction at certain other stages in the computation possibly without affecting the end result, there would be an impact on the data quality as measured by the differencing metric which is undesirable. Quality preservation has been used as the method of determining which computations to perform.

4.13.3 A Note on Order

Measurements have to be both smoothed and clock-corrected. In which order should these computations be performed? Let S represent the smoothing operation and T the time correction operation. Which of the following expressions is the most appropriate for a smoothed time-corrected pseudor-

ange?

1. $P' = T(S(P, C))$

2. $P' = S(T(P), T(C))$

The initial thought was that 2 was correct from the following line of argument. Due to the excessive clock drift on the Garmin receiver, both P and C would increase with time even if the satellite were stationary. So two values of P, say, measured at different times *i.e.* $P(t_1)$ and $P(t_2)$ are like apples and oranges *i.e.* almost measured in different units of distance. To combine these meaningfully in an algorithm, requires them to be corrected to the same unit. However, time correction of P on its own, was proven to give crude results so some suspicion is associated with this option. Also, the dual time correction necessary before proceeding any further provides a complex source of error.

Further consideration was that 1 is correct. One could think of P and C simply as quantities that drift smoothly upwards on their own (the fact that it is due to clock error is simply immaterial) and a smoothing algorithm should still work. Clock correction based on a single smoothed quality is simpler and has fewer sources of error. However, the conclusion is tentative.

4.14 Intermediate Results

DIY Differential GPS with No Smoothing 95 measurements			
coordinate	value	range	variance
X	3945578.69 $\pm$ 0.08 m	1.75 m	0.141 m
Y	65494.51 $\pm$ 0.06 m	1.24 m	0.095 m
Z	4994177.74 $\pm$ 0.05 m	1.35 m	0.059 m
Average distance from average position		=	0.50 m
Maximum distance from average position		=	1.05 m

DIY Differential GPS with Carrier Phase Smoothing 95 measurements			
coordinate	value	range	variance
X	3945578.70 $\pm$ 0.06 m	1.15 m	0.093 m
Y	65494.50 $\pm$ 0.06 m	0.98 m	0.078 m
Z	4994177.72 $\pm$ 0.05 m	0.82 m	0.059 m
Average distance from average position		=	0.42 m
Maximum distance from average position		=	0.74 m

Firmware Non-Differential GPS (non-DIY) 618 measurements			
coordinate	value	range	variance
X	3945323.47 ± 0.17 m	12.38 m	4.583 m
Y	65219.48 ± 0.11 m	5.88 m	1.890 m
Z	4994367.98 ± 0.08 m	4.38 m	1.112 m
Average distance from average position		=	2.40 m
Maximum distance from average position		=	10.04 m

DIY Non-Differential GPS 6090 measurements			
coordinate	value	range	variance
X	3945323.60 ± 0.06 m	12.52 m	4.630 m
Y	65219.55 ± 0.04 m	7.00 m	2.517 m
Z	4994368.56 ± 0.03 m	13.12 m	1.098 m
Average distance from average position		=	2.57 m
Maximum distance from average position		=	13.06 m

These are some representative first positional results from using the replacement Garmin G12XL hardware. Though not spectacular, some benefits in accuracy do finally result from using carrier phase smoothing. The accuracy of single receiver DIY calculations mirrors that of the firmware built into the receiver. Overall these results confirm the correct integration of the Garmin unit within the GPS system.

4.15 Other Software

To check the functioning of diygps, a short exercise was undertaken to examine the comparative accuracy of equivalent software. The only suitable software is commercial and only operates in a post-processing manner. Gringo/P4 (below) is the only commercial program which can talk to the G12 directly, and this was the main avenue of investigation. Gringo/P4 requires ephemeris data which is only available off line, so again it operated is a post-processing manner.

4.15.1 Gringo + P4

Gringo (GPS RINEX Generator) is a Windows program which records the raw receiver measurements (pseudoranges and, optionally, carrier phase measurements) from 12-channel Garmin GPS receivers in RINEX format. RINEX stands for Receiver Independent EXchange Format. The software is commercially available from the University of Nottingham and first appeared on the Web around the time that the Garmin “secret” protocols for raw data were cracked.

P4 (Pseudorange and Phase Post-Processor) is the associated Window program which postprocesses the RINEX files to provide a position fix. Three input files are required for a differential fix: a RINEX file for the reference receiver; a RINEX file for the mobile receiver; and a satellite "ephemeris" file. Clearly, a differential position fix will only be available for the intersection of the time periods during which the individual files were recorded. The two RINEX files can be produced by Gringo and the ephemeris file is available on the Web (http://www.ngs.noaa.gov/CORS/Data.html).

The documentation for Gringo claims that "centimetric positioning" is possible under the right conditions. However, practical tests with Gringo revealed that it achieved no more accuracy than "diygps" — indeed probably less and the differential solution was observed to jump around by distances of the order of a metre. This test confirmed that "diygps" was probably operating at the appropriate level for the hardware and solution techniques employed. Indeed, all the authorities on GPS consulted over the Web felt that 1.8 metres was a good medium-term accuracy: even Chris Hill who is the author of Gringo and p4 held the same opinion.

4.15.2 Batch GPS Processing on the Internet

It is possible to submit one's GPS data to a website, and to obtain post-processed cm-level positioning anywhere in the world.

- GPS Solutions Inc

  http://www.gps-solutions.com/

- Auto Gipsy Precise Point Positioning Service

  http://www.unavco.ucar.edu/processing/gipsy/auto_gipsy_info.html

- SOPAC Coordinates Generator

  http://sopac.ucsd.edu/cgi-bin/coordinatesGenerator.cgi

- DCI Wide Area DGPS Correction Service

  http://www.dgps.com/inversemail.asp

However there are some caveats: the data submitted must be dual frequency and the receiver must be static. Essentially, this is no different to running a batch job, and indeed the services are based upon commercial high precision GPS software. The website handles the retrieval of relevant GPS data archived on the Web.

The service offered by GPS Solutions Inc is based upon the high precision Bernese GPS software; must be paid for; and the results come back by email the following day. A service that could operate real-time for free would clearly be far more useful for AR. In fact, there are a number of free GPS processing services on the Internet, including the other three listed, but again these are batch oriented (results may be turned around in 5 minutes) and intended for high-end GPS equipment.

4.15.3 Internet-Based Global Differential GPS (IGDG)

In contrast, the most fully developed global real-time differential service on the Internet is IGDG. This is commercial, operated and maintained by JPL, using a subset of 12 stations of the NASA Global GPS Network (GGN). A local dual frequency receiver is required. GGN consists of 60 sites which have traditionally been operated in batch mode. IGDG calculates accurate ephemeris and clock corrections, and these are sent over Internet as corrections to GPS broadcast ephemerides together with differential corrections. IGDG promotes itself as "by far the most accurate real-time global positioning system", and claims 10 cm horizontal and 20 cm vertical real-time positioning accuracy.

4.15.4 Virtual Reference Stations

Differential GPS and particularly Real-Time Kinematic (RTK) positioning is limited by the baseline distance. To cover a region with such services would require a dense (and hence expensive) network of base stations. The concept of a virtual reference station is an obvious solution, where interpolation occurs between physical base stations to produce readings as if they came from a close virtual base station. Spectra Precision Terrasat provide such a product: continuous modem connections are required between the control centre and all reference stations. The control centre calculates optimised RTCM message for each user. Spectra Precision Terrasat have virtual networks which span regions of Europe and Japan. Their demonstration Web software to download a virtual reference station data file did not work, though it is possible to download physical reference station data files.

4.16 High Accuracy GPS by Ambiguity Resolution

4.16.1 The Requirement

Through the tests carried out, and the direct observation of cycle slips causing jumps in otherwise smoothly varying carrier-smoothed pseudoranges it was realised that to advance significantly the ac-

curacy of diygps in software terms (if indeed this is possible with the quality of data available), the next step is to resolve the integer ambiguity problem. This issue of ambiguity resolution is still at the research phase. A bibliography was located for GPS Ambiguity Resolution [54] while current papers are to be found in the proceedings of the related Institute of Navigation conferences *e.g.* ION GPS 2000 [55]. There are but fragmentary amounts of open source ambiguity resolution code on the Internet:

- Mathematical Geodesy and Positioning (MGP), Delft University of Technology

  Note software is only available on request rather than by downloading.

  `http://www.geo.tudelft.nl/mgp/lambda/software.html`

- Ambiguity Decorrelation algorithm by Shaowei Han

  `http://www.ngs.noaa.gov/gps-toolbox/han-04.htm`

- MATLAB code by Kai Borre

  `http://www.i4.auc.dk/borre/download.htm`

The majority of ambiguity resolutions systems work with dual frequency data. The dual frequency problem is clearly easier than the single frequency one. In addition, the data from these dual-frequency "geodetic-quality receivers" is generally higher quality. It would be useful to take the results of the academic research into a practical public domain code, that provides the highest accuracy position possible (through ambiguity resolution) to users of cheap single-frequency handhelds like the Garmin G12XL. Work was carried out based on two of the public domain LAMBDA codes, one originating from the University of Delft [51] and the other from a GPS textbook [45].

4.16.2 University of Delft Software

Introduction

The FORTRAN code was requested from the Delft University of Technology and tested. The executable is called *lambdabe*. The inputs are :

$\boldsymbol{n}$ the number of independent double difference measurements

This is the product of the number of satellites in view and the number of GPS frequencies used.

For example, with 6 satellites in view and range measurements at both L1 and L2 frequencies, n is 12.

$\boldsymbol{a}$ the float solution

This is an n element vector, of the pseudoranges expressed in wavelengths.

$\boldsymbol{Q}$ the variance-covariance matrix of the double differences

This is as n by n matrix. However, as Q is symmetric, only the lower triangular parts are used from the supplied linear array of n^2 elements. For example, if n were 3 then only elements 1,2,3,5,6 and 9 are used.

The outputs are :

$\boldsymbol{a'}$ the fixed solution This is an n element vector, which contains the integer ambiguities in the input pseudoranges.

$\boldsymbol{disall}$ squared norms sorted in increasing order

This length of this vector is $MaxCan$ which represents the maximum number of candidates, whose norms are reported back to the caller in this form. $MaxCan$ is a constant in the code - currently set to 2. Looking at the size of norms of the the first and second candidates gives an indication of how clear a winner the first candidate is. This acts as a reliability measure for the solution in a'.

lambdabe provides a choice between two different versions of the solution code `LAMBDA` and `LAMBDA4`. Both return the same answer, but `LAMBDA4` is later and better code, and was used for all subsequent work after checking their equivalence. `LAMBDA4` efficiently calculates the value of Z'^{-1} directly rather than calculating Z (one of the internal matrices used) first.

Software Integration

The code was ported from FORTRAN to 'C' using the `f2c` conversion software, and verified using the test files and data values supplied. For portability and stringency the code was further changed so it could be compiled by a C++ compiler without any warning messages, and then it was re-verified.

The next stage was to integrate the LAMBDA code with the real-time GPS system. Double differences were computed as were triple differences for completeness, and the variance-covariance matrix was computed. However, the results that came out of the exercise were not good, and it was realised that the integration had probably been carried out incorrectly. In particular, there was little confidence in the way the variance covariance matrix had been computed. The LAMBDA code as supplied, was of limited use as it did not show how the variance-covariance matrix was generated. Fortunately some other LAMBDA code was located on the web, which takes lower level satellite data as the starting point.

4.16.3 Teaching Software

Introduction

This LAMBDA software takes the form of pedagogic MATLAB code taken from the book "Linear Algebra, Geodesy, and GPS" [45]. The top-level procedure is called `proc_dd`, and has three input files as parameters. For example, a sample invocation is:-

```
proc_dd('SITE1.96O', 'SITE2.96O', 'SITE1.NAV')
```

where:

SITE1.96O	-	GPS range observations from first site
SITE2.96O	-	GPS range observations from second site
SITE1.NAV	-	ephemeris data over observation period

These GPS range observation files are in standard RINEX format, whereas the ephemeris data are in a simple binary format file, that is readable by MATLAB. The output of this code is the vector from the first to the second receiver *i.e.* the method returns a relative position and one that is hopefully of high accuracy, For example:

```
Vector from 1 to 2:  -225.781 -461.434 192.874
```

Software Porting

The code was ported to octave, so it could run under Linux. Octave is a public domain MATLAB clone. Needless to say, the port of around 3000 lines represented a considerable amount of work and utility routines had to be written to match the facilities of MATLAB absent in octave. These routines

were individually unit-tested, as their function was not entirely clear from their documentation. Syntax changes were also required. The port to octave was verified. One of the problems of debugging the software is that octave has no type-checking. This is especially problematic if the runtime is long *e.g.* the application takes a minute or so to run with the example files supplied. So the code may fall over tiresomely at a simple type error after 50 seconds, which would have been picked up immediately by a system with type checking, as the code was loaded rather than when executed.

The code was finally ported to 'C' in preparation for integrating with the real-time GPS system. Clearly, an automatic port would have been preferred and the manual port was a last resort. MATLAB under Windows has a facility to output code in 'C', but this is a commercial add-on, and none of the MATLAB installations investigated had it. The port of 3,000 lines of MATLAB to 'C' took the best part of the fortnight. 5000 lines of 'C' resulted, as extra code had to be provided to support the MATLAB facilities. The ported code naturally did not work immediately. Debugging the code was also a considerable effort, as at each stage output from the 'C' and MATLAB codes had to be compared. Eventually, the 'C' produced the same output as the MATLAB code. The resulting 'C' code runs in under one second (*cf.* 1 minute for the octave version) which allowed subsequent experimentation on the code to have more immediate feedback.

Batch to Real-time Conversion

The didactic code is batch processing and provides a single position fix at the end of execution. The software was converted to work with a sliding window of data, to provide the necessary period of historic integration for a dynamic sequence of position fixes. The number of measurements in the sliding window is a run-time parameter, and this was varied to study the effect on accuracy. A minimum of three measurements must be used for the mathematics of the method to work, and unsurprisingly using less than three measurements gives results which are nonsensical. For the input data file used, 33 range measurements were taken in total at 30 second intervals. The zero position is taken to be the position calculated using the LAMBDA method with the maximum 33 range measurements. The other results are presented relative to this. For an integration period of 32 measurements only two position fixes are possible, whereas for periods of 4 measurements 30 position fixes are possible, *etc.* .

The graph in Figure 4.18 shows how the accuracy of the LAMBDA method increases with the number of measurements used. The results are particularly dramatic, showing the potential accuracy

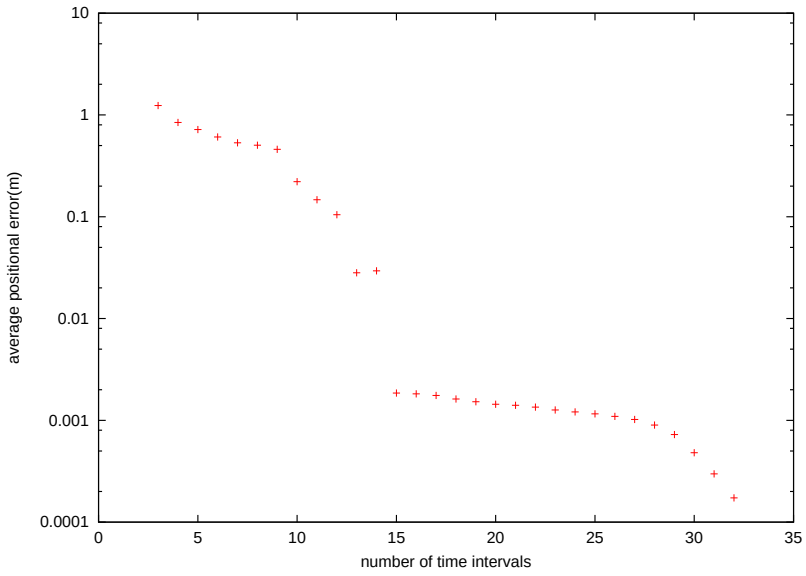

Figure 4.18: LAMBDA method accuracy against number of measurements

of the LAMBDA method: moving from 1 m accuracy with 3 measurements to 0.2 mm accuracy with 32 measurements. There are two marked rapid improvements in accuracy at around 15 measurements and again at around 30 measurements. The first improvement to around 2 mm is very sharp - especially taking into account the logarithmic scale in use. The second improvement to around 0.2 mm corresponds to a distinct and clearly significant gradient change in the logarithmic graph.

The graph is open-ended in the sense that the maximum accuracy in the LAMBDA method has not been reached. Were extra readings to be taken then extra accuracy would likely have been obtained. However, this accuracy is not much more significant than the fact that an average measure of a quantity converges for ever. The issue is really the rapidity of this convergence for convenience of operation and indeed particularly within an interactive AR context. The graph shows that only 15 measurements are required for the accuracy to be well within that required for AR. It is open question whether the reliable result is because 15 measurements were taken (statistical sample size dependence) or because these measurements were spread across a 7 minute period (temporal dependence). The data could be analysed further to determine whether the same number of measurements spread over a longer rather than shorter time interval gives better results or not. This analysis is critical for the use of the LAMBDA method in AR. If the dependence is largely numeric then this is an argument for GPS units to operate at high frequencies - moving from the current maximum of about 10 measurements per

second to say 100 Hz. At 100 Hz the latency for a highly accurate position measurement would then be 0.14 s. There is no technical reason why GPS cannot operate at significantly higher frequencies than at present. If the dependence is temporal, then the LAMBDA method may be inappropriate for AR.

Of course, it may be possible to vary the nature of LAMBDA calculation to make the position calculation more dynamic. For example, a longer integration period could be used to produce the variance covariance matrix compared to a shorter period used to determine the float solution. Indeed, the University of Delft code was integrated is such a way that the software permitted different integration periods for Q and a. However, it was not established whether this was a valid approach mathematically.

Software Integration

To start to use the pedagogic LAMBDA code with real data coming from our G12 GPS receivers there are two possibilities:

1. Incorporate the LAMBDA computational kernel within the real-time GPS system
2. Adapt the real-time GPS system to produce RINEX files

Approach 2 was chosen because it was the simpler and involved the least effort. Developing the LAMBDA code further was going to be simpler with a small executable, rather than as part of a larger system. Eventually, when the LAMBDA code had been taken to completion and tested in isolation, it could be incorporated into the real-time system. However, approach 2 contains a trap for the unwary. Converting real-time GPS data (time is given as a second within a week) into a RINEX file (time is given in a full date format) is consequently **not** a well defined problem. The unexpected complexity of getting round this problem is explained in Appendix C.

4.16.4 Cycle Slips

Supplied with the MATLAB LAMBDA code are data files which contain cycle slips. The ported code is run from the command line thus (`dod` stands for DOuble Difference):

```
dod 00050761.94O 08100761.94O FJELLERA.NAV 20
```

where:

00050761.94O	-	GPS range observations file from first site
08100761.94O	-	GPS range observations file from second site
FJELLERA.NAV	-	ephemeris data file over observation period
20	-	maximum no of measurements for integration (dynamic system will use whatever number of measurements is available if this is < 20)

With these data sets the LAMBDA code no longer computes an accurate position. For a certain period the reported relative position will be stable: then a presumed cycles slip occurs and the position becomes non-sensible Then when this slip falls outside the integration period the reported position is again stable, but observably not quite at the same location as before. The LAMBDA software as it stands is not capable of detecting or responding appropriately to these cycle slips. There are two means of responding to a cycle slip:

1. excise data with cycle slips from calculation

   This is the simplest approach, but impacts on the accuracy of the overall calculation. There are many ways in which the excision could be performed. The most simple-minded are on a satellite or temporal basis, but clearly more intricate methods could be employed.

2. attempt to correct cycle slip

   This is the more sophisticated approach that requires more intelligent software. The benefit is the potential gain in accuracy, the downside is the possibility of catastrophic failure if the correction is wrong. There may well be enough redundancy in the satellite data to resolve the majority of cycle slips, but the use of another positioning technology could be of enormous benefit here.

An initial attempt was made to detect and respond through excision to cycle slips in the earlier integration exercise when the University of Delft code was incorporated into the real-time GPS kernel. The double and triple differences were computed. The triple differences of the phase showed an impressive accuracy on the whole, and an atypical jump in this quantity to a larger value was a clear indication of a cycle slip. However, at that stage it was less clear how to deal with a cycle slip given that fact that there was little confidence in whether the LAMBDA method was being called correctly in the first place. Indeed this was why work with this software was abandoned in favour of work with the teaching software

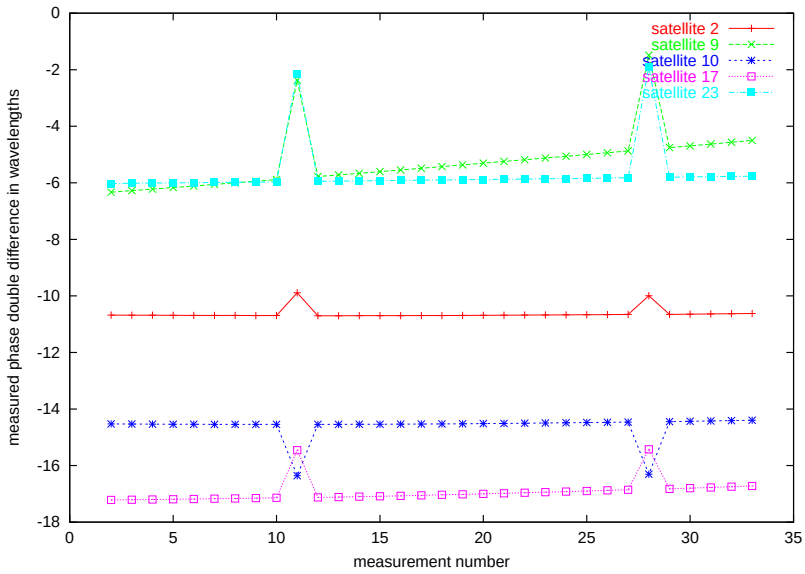

Figure 4.19: Double difference — ideal sample data

This time around the LAMBDA method was known to work, so the next step was to detect a cycle slip through triple differencing (easy as this was accomplished earlier) and then to move on to compensating for this cycle slip. Unfortunately, this is when the fortnight allowed for high-accuracy investigation ran out! Note that working differential GPS hardware was not obtained until after the write-up time for this thesis. A fortnight was subsequently taken out of the write-up period to look at the high accuracy techniques which were identified as important.

The double difference graph for the sample data files supposedly free of cycle slips is shown in Figure 4.19. In fact, two cycle slips are observable, but these somewhat surprisingly did not greatly affect the operation of the LAMBDA method. Detecting and indeed correcting the cycle slips in this scenario where a cycle slip is an occasional one-off event would be trivial. A cycle slip corresponds to a significant proportion of wavelength, and it is trivial to find an appropriate threshold, given the size of the cycle slip compared to the size of the noise in the double difference when no cycle slips occur. The double difference should be a constant, but this is observed to drift slowly in an approximately linear fashion. The cycle slips occur across all satellites when they happen.

The double difference graph in Figure 4.20 for the sample data files which officially contain cycle slips does indeed cycle slip with the same characteristics as before. Note the situation is further complicated as satellite 17 becomes invisible. Later on satellites 7 and 12 become visible. Finally

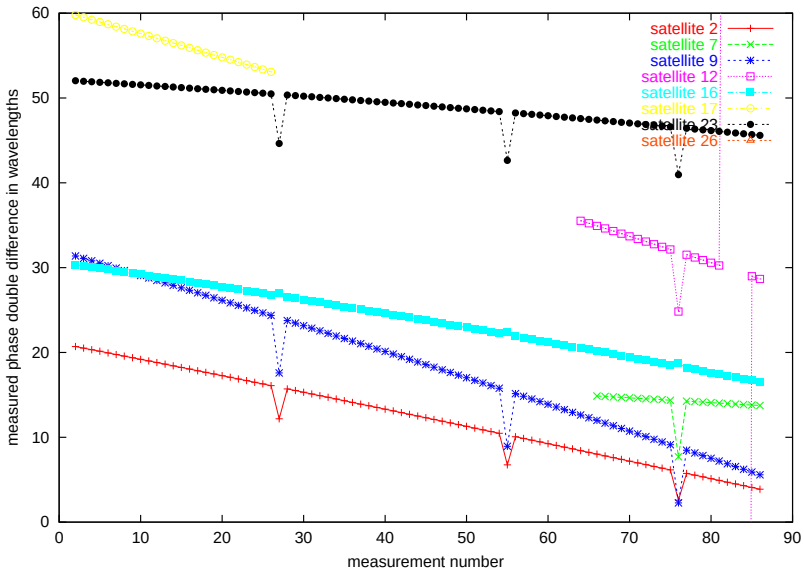

Figure 4.20: Double difference — problematic sample data

satellite 12 suffers a huge cycle slip, and drops out of sight for an instant. The y-axis of the graph was deliberately chosen not to be swayed by the size of the peak. Presumably the signal from satellite 12 is weak. Conversely, the signal from satellite 16 is presumed to be strong, as the cycle slips are just discernable above the noise by eye. However, numerically they are still easily detectable. The disappearance and emergence of satellites makes this a good test file for making the LAMBDA method more robust, a process which should be carried out.

The double difference graph for data collected live off the G12 units is shown in Figure 4.21 and has different characteristics. For a start the double difference does not appear to drift from zero. However, it has to be remembered that the readings were taken at one per second, compared to once every 30 seconds for the previous two graphs. There are no cycle slips as the value of the double difference does not jump a significant proportion of a wavelength. The maximum change observed is about 15% of a wavelength. Moreover there is, at times, a curious sign-changing aspect to this error, which is likely to be a anomalous receiver artifact and could be smoothed out. At other times when this sign changing is not occurring, the error is very small indeed. Satellite 8 keeps drifting in and out of the picture, making the situation very messy indeed for the LAMBDA method.

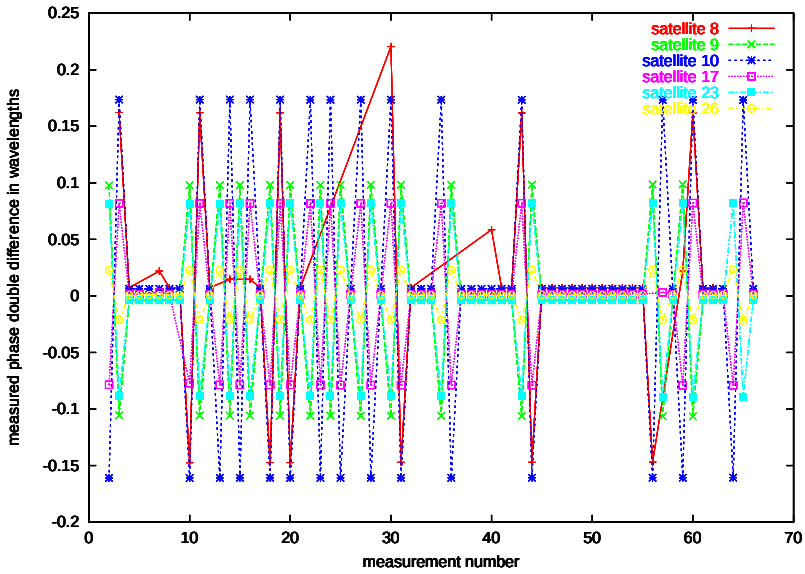

Figure 4.21: Double difference — live data from G12 H/W

4.16.5 Ambiguity Resolution Conclusions

Adapting the pedagogic LAMBDA software for real world data with cycle slips and fickle satellites is not so much technically challenging, as just very fiddly. There does appear to be sufficient redundancy in the data to easily detect and correct cycle slips, when these occur. However, dealing with satellites which can disappear and re-appear at any time is problematic.

The accuracy the double and triple difference phase measurements exhibit, taken both from the sample data files and from our hardware units, shows their usefulness in dealing with cycle slips. The lesson learned from this exercise is that it is always a good idea to supply exemplar “from raw” data to test out numeric computation code. Otherwise, the software developer is uncertain whether any problems lie in the code itself or in the quality of data being obtained from equipment, This is why the didactic code was far more useful in practice than the code from the University of Delft: sample data files were supplied.

A next step is to work with the data files which are known to incorporate cycle slips but are of the same quality as the earlier test files used which were free from cycle slips (shown in Figure 4.20). An alternative next step is to use data from our own equipment gathered in open sky conditions, such as at Gosbecks, which should result in a suitably long window for analysis where a fixed set of satellites are consistently available. Then it should be possible to make some kind of assessment

of whether the data quality from the G12 units is up to the LAMBDA method or not. The data gathered from our GPS equipment (shown in Figure 4.21) and used for analysis was actually taken from the roof of the University of Essex, due to lack of equipment portability. The site used for taking measurements only had a partial view of the sky, and undoubtedly there was some multipath resulting from reflections of signals off campus buildings. Finally, once a stable software platform has been developed, a comparative evaluation of both the LAMBDA codes with real world data would be useful, either to establish a winner or as mutual verification.

4.17 Relative GPS

Relative GPS can be the simplest differential GPS technique. For this to be the case, the displacement of a mobile GPS receiver's true location from its measured GPS position, is taken to be the same as the displacement of a second reference unit's true position from its measured GPS location. As with conventional DGPS, in order for this technique to work the reference receiver must remain at a known location *i.e.* one whose position has been surveyed accurately. There are two additional caveats, however:

1. the two receivers must be close (*i.e.* within around 15km)

2. the identical set of satellites is used for calculations at each sites.

To meet condition 2 there must be some form of coordination between the two sites, and this is clearly impossible to ask of a pair of autonomous GPS handheld units, each of which works in a "standalone" manner. To ensure coordination, the calculations for both receivers are performed by the diygps software, which was suitably enhanced to perform relative GPS calculations ensuring only the satellites in view at both locations are used. It was felt important just to check out this naïve technique for completeness, as it represented very little additional software effort.

4.18 Final Results

Once the modifications for relative GPS has been made to the diygps program, it was felt a suitable time to review the accuracy of single and dual receiver GPS experimentally, using only low accuracy differential techniques. All techniques were used which return an ECEF or (X,Y,Z) result. Each analysis is based on the same set of input files, which originated from a relatively long (half-hour)

recording of the RS232 raw-data outputs from two Garmin G12 receivers and one Marconi AllStar receiver. One measurement was taken every second. The different GPS variants studied are shown in the following table:

Case	Receiver	Technique	Smoothing
1	first Garmin G12	DIY standalone	no
2	second Garmin G12	DIY standalone	no
3	Marconi Allstar	DIY standalone	no
4	Garmin G12 $\times$ 2	DIY differential †	no
5	Garmin G12 $\times$ 2	DIY differential † (time corrected)	no
6	Garmin G12 $\times$ 2	DIY differential †	yes
7	Garmin G12 $\times$ 2	DIY differential † (time corrected)	yes
8	Garmin G12 $\times$ 2	DIY relative	no
9	Garmin G12 $\times$ 2	DIY relative	yes
10	Garmin G12 $\times$ 2	DIY differential	no
11	Garmin G12 $\times$ 2	DIY differential	yes
12	Marconi Allstar	proprietary firmware	no

Though cases 4 and 6 may appear to be identical with cases 10 and 11 respectively, the code for cases 4 through 7 (as indicated by †) has been considerably rationalised and enhanced to be robust to the absence/presence of receiver clock corrections. Note that the starting-point GPS code used would not work correctly with Type 3 units if clock corrections were not applied, and it is a reassuring check of the various versions of the software, that the results in cases 4 and 10, and in cases 7 and 11 are identical. In fact, the time-corrected results themselves are no different from the non time-corrected results, functionally rendering cases 4 and 5 equivalent to cases 6 and 7 respectively. Most significantly, the results from using relative GPS and our initial differential GPS method are identical, both with and without carrier phase smoothing. This is as one might expect, but is surprising in that the computational methods employed are very different. The accuracy results are presented in the following table:

Case Equivalences	Receiver	Technique	Average Error	Max Error
1	G12	DIY standalone	9.00	18.28
2	G12	DIY standalone	8.22	32.33
3	AllStar	DIY standalone	8.71	14.55
4, 5, 8, 10	G12 $\times$ 2	DIY differential	7.74	42.54
6, 7, 9, 11	G12 $\times$ 2	DIY smoothed differential	4.76	27.07
12	Allstar	proprietary firmware	8.35	10.20

Figure 4.22 compares the standalone DIY software against the standalone proprietary firmware for the AllStar unit. Modulo a very small constant offset these positional results have the same distribution

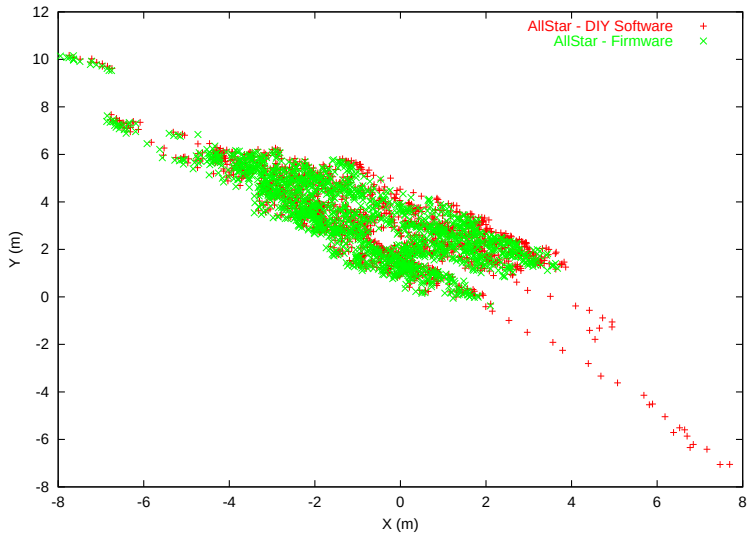

Figure 4.22: AllStar results — DIY processing *vs.* firmware

pattern, and are essentially identical giving us confidence in the DIY algorithm. The DIY results have a few additional rogue points, but these are probably the less accurate positions that are computed while the diygps software builds up initial satellite information: whereas the firmware in the receiver itself will already have this information.

Figure 4.23 shows, the increasing accuracy of moving from standalone GPS to differential GPS to carrier-phase smoothed differential GPS. The average error reduces from 9.0 m through 7.7 m to 4.8 m, which can hardly be called spectacular though the results are at least in the order expected. Phased-smoothed differential GPS only improves accuracy by a factor of two. If SA had still been switched on, then differential GPS would have made a more distinct difference! It is worthwhile pointing out that GPS accuracy over shorter time periods is greater and that this is exploitable by AR. The half-hour sampling period is essentially the “infinite” sampling period for collecting as wide a range of GPS positional measurements as possible. Although most types of error will be seen in this period, some references say that safety can only be “assured” after a fortnight of data collection to ensure that a significantly wide range of satellite configurations and atmospheric conditions have been encountered.

It may be possible that the use of different hardware could improve results: we were reduced to interpreting undocumented RS232 protocols from consumer-grade equipment to obtain a working system. However, to obtain significant accuracy improvements over a single receiver, it probably

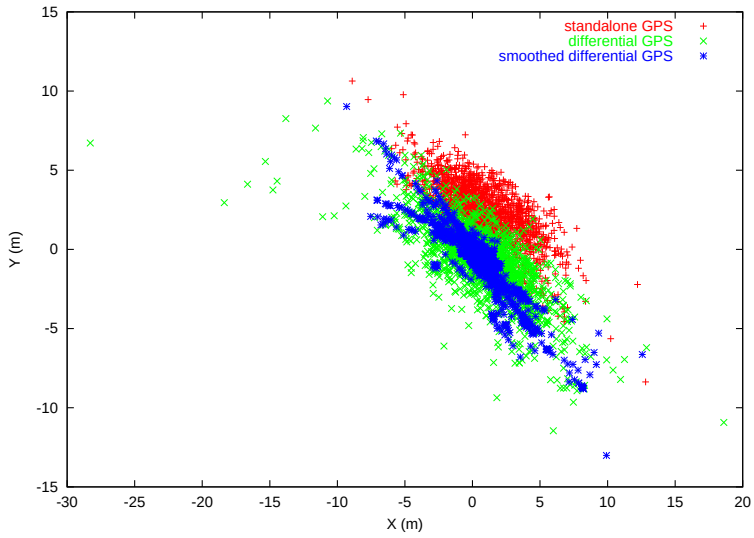

Figure 4.23: G12 results —- standalone *vs.* differential *vs.* smoothed differential

essential that a totally different differential GPS algorithm geared for high accuracy (such as the LAMBDA method) is used.

Figure 4.24 shows the difference in position distributions for two identical stationary Garmin G12 GPS receivers. The reported positions from the second receiver are more tightly clustered, but the maximum error is greater. The difference is possibly due to the fact that the external antennae used by the receivers are different models. This trend of lower average error, yet greater maximum error is apparent in much of the GPS processing above. A more advanced technique may produce a significantly lower error overall, but there is still the possibility of the system falling apart more catastrophically. It is important that the advanced GPS processing techniques are used under favourable circumstances, with good quality hardware, error checking software and favourable satellite visibility. Furthermore, measurements taken at both L1 and L2 frequencies should be available if possible. Fortunately, at Gosbecks good satellite visibility is guaranteed, and the only variable is the quality of the hardware. Some objective metric of hardware quality should be proposed, so that minimum requirements for AR-grade accuracy can be established. Such a metric could be based on, say, the quality of a triple difference measure rather than positional accuracy, which has such a dependence on satellite geometry. Where a pair of satellites is in constant view, there is no reason why the majority of double difference cycle-slips should occur.

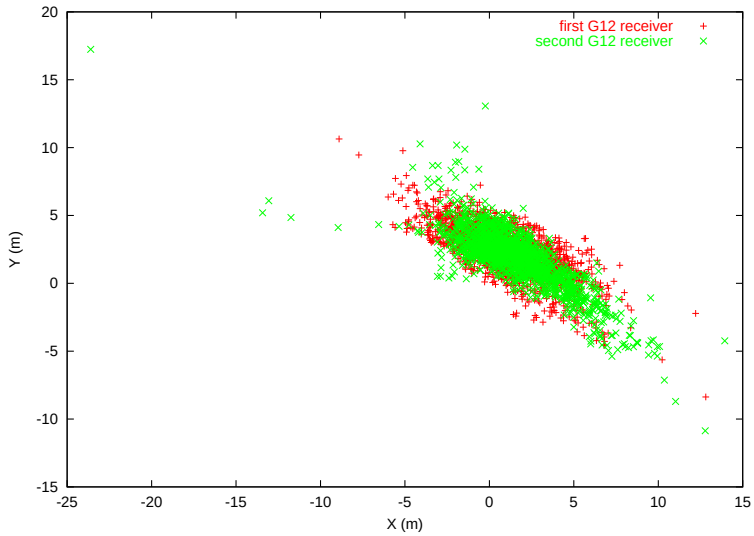

Figure 4.24: G12 results — first *vs.* second unit

4.19 Slip Threshold

In the smoothing algorithm a threshold of 15m is used to determine whether a cycle slip has occurred. This is actually a very crude test, but at least it is simple and robust. On the other hand while the triple difference measure is a better way of determining cycle slips, an intelligent interpretation of this measure may get it wrong!

Initial plots of the differential GPS positions described a linear feature rather than the expected elliptical splodge. This suggested that the position was wandering off consistently but probably unnecessarily, due to cycle slippage. To this end, the effect of varying the threshold was studied. A comment in the original code indicates that the supplied figure of 15 m is very much guess work. The results of this study, are shown in Table 4.17 for the smoothed differential GPS position. Figure 4.25 shows the plot of this table of data *i.e.* average error against cycle slip threshold. As the threshold tends to zero the accuracy tends to that of non-smoothed differential GPS. By reducing the threshold from 15 m to 8 m, the error is halved and the position plot no longer takes on the form of a straight line. The most effective threshold was found to be 2.6 m, but it is not particularly critical as long as it is in the range 2 to 8 m — a wide landing strip! Within the range 2 to 8 metres there appears to be two principal and distinct local minima; the author cannot explain this. The number of points on the curve had to be limited as each one represents a complete reprocessing of half-an-hour's worth of GPS

cycle clip threshold (m)	average error (m)	maximum error (m)
1500	24.2254	42.4648
150	18.2842	66.9628
15	8.8825	38.5347
8	5.2600	26.8477
7	4.9534	26.9412
6.5	4.9926	26.9203
6	4.8727	26.9601
5.5	4.9868	26.8991
5	5.0237	26.9145
4	4.9602	26.9366
3	4.8478	27.0536
2.75	4.8487	27.0625
2.6	4.7601	27.0744
2.5	4.7769	27.1133
2.4	4.7860	27.1647
2.25	4.8677	27.2425
2	4.8910	32.6314
1.5	5.4891	31.1693
1	6.6731	30.4695

Table 4.17: Effect of changing the cycle slip threshold

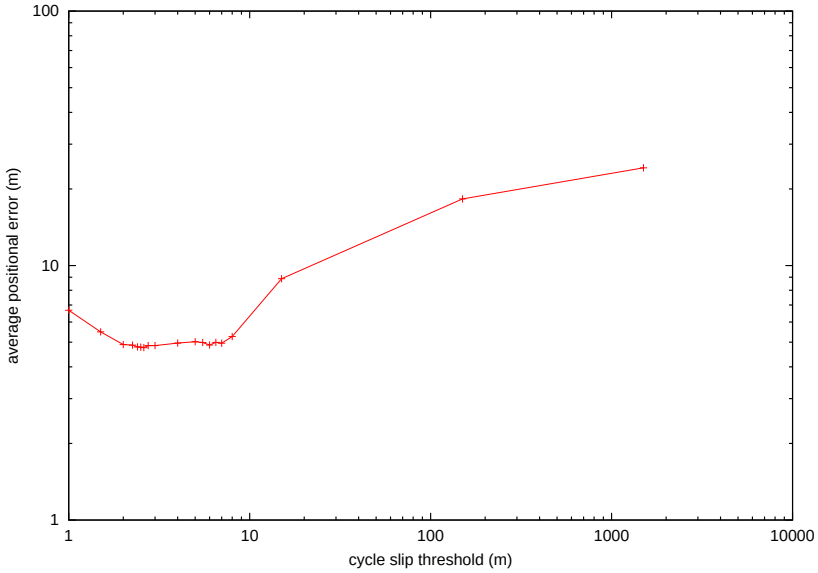

Figure 4.25: Average positional error *vs.* slip threshold

data. Each run takes a considerable time although it can be done somewhat faster than real-time. This optimal threshold is undoubtedly hardware dependent on the characteristics of the G12 GPS receiver. A new optimal threshold would have to be established for each type of receiver used.

4.20 Conclusions

Four qualitatively distinct, self-contained classes of GPS system were built: non-differential; symmetric differential; asymmetric differential using Internet-based corrections; and integer ambiguity resolved differential. Their unifying distinguishing features are a combination of low-cost hardware and in-house developed, open source code. The best accuracy achieved with non-integer ambiguity resolved differential GPS is only twice that of non-differential GPS. Although this improvement is useful, the requirements for AR are still not well-addressed. The limiting factor in the accuracy of this technique was shown to be the quality of the hardware. Note that if the GPS signal degradation had not been switched off by the US government during the project, then the relative improvement of differential GPS would have been much more significant.

The enhanced processing technique of integer ambiguity resolution was shown to give astonishingly good accuracy, more than sufficient for AR, but these results were achieved with archived data files rather than measurements taken directly from our equipment: time for all practical experimentation was strictly limited as working differential equipment was only obtained after the end of the project. Nevertheless, consumer grade GPS units (to replace the faulty dedicated OEM boards purchased) bought at the last minute when details of how to crack their secret protocol were published on the Internet were turned into a practical different GPS system — a task for which they were never intended. One of the most significant capabilities of the asymmetric system, is the new ability to turn the inexpensive Garmin G12 handheld into a (traditionally expensive) device supplying a differential GPS correction.

For GPS to be suitable as a positioning technology for AR currently requires bringing in a secondary position technology for sensor-hybridisation. Further work is required to show whether a positioning system based on GPS and integer ambiguity resolution processing alone is suitable for a highly dynamic AR application: the static accuracy of the technique is certainly more than sufficient. Of course, there is a total dependency on the quality of the raw data emerging from the GPS units and all conclusions are subject to revision on the basis of future products in the consumer segment of the

market that has been addressed by this work. The Internet is currently underexploited as a vehicle for delivering GPS connectivity. One of the systems built is Internet-based, while the others are network compatible, and a proposal for a global Internet-based GPS system was presented which could supply significantly increased accuracy to all GPS users.

Chapter 5

A Computer Vision System for Augmented Reality

5.1 Introduction

Vision is appealing as a positioning technology because of its inherent transparency of operation. The principles have been long established within the fields of surveying and photogrammetry. Indeed, vision has a unique rôle to play in AR, because any solution to the registration problem will be judged ultimately in terms of a visual pathway. The vision-based position systems for AR that exist (surveyed in Chapter 2) are based on a few specially designed targets [56] or on large collections of points of light [31] which simplify the computer vision processing required. Through establishing the positions of identifiable scene objects within an image, it is possible to derive the position and orientation of the camera to enable correct overlaying of the virtual on the real. Within the special effects industry solving the registration problem is also hugely important. However, the technicians have the luxury of solving what they call the "motion match problem" off-line. Complete automation is not required and even simplistic object tracking algorithms are useful, as manual intervention can supply both the initial positions of the objects within the scene and then correct any subsequent mis-tracking.

The literature concentrates on the geometry of the problem but largely ignores the initial image processing which is critical for success within AR. This chapter describes the image processing techniques that were developed to produce a necessarily robust vision-based positioning system for AR. The geometric aspects of the system are presented in the next chapter. The majority of the effort was expended on the vision processing side: the results of experimenting with different target designs and

detection algorithms to achieve reliability are reported in the following sections.

5.2 Choices

This section discusses the surprising number of subtle choices have to be made before a vision system can be developed.

5.2.1 Use Targets or Unmodified Scene?

Should the real-world environment intentionally be altered to decrease the complexity of the image processing task? Edges, for example, can be more easily located if they are painted white. In fact this "alteration" is done to roads to assist the task of human vision while driving. The least invasive and most generally practiced environmental alteration is the mounting of specially-designed targets or fiducials (*fiducia* is the Latin for trust) within a scene. Clearly, the flat planar surfaces of an indoor environment are more amenable to this form of tagging than an exterior environment like the Gosbecks Archaeological Park.

Initially, a vision system was rejected for two reasons. Firstly it is not feasible to cover the Gosbecks Archaeological Park with targets that will be visible everywhere; ideally an exterior AR system should work in an arbitrary environment. Secondly there is the consequent difficulty of automatically interpreting an un-targeted scene and then correlating this with a computer model to establish location. Although, a virtual representation may exist, this may not be in a suitable form for correlation. For example, the building in the scene may be a ruin but the virtual model may be a reconstruction. Indeed if the AR model were perfect for correlation, it would beg the question of why the reconstruction was being attempted in the first place!

5.2.2 Manual, Automatic or Surveyed Registration?

There are three ways in which initial registration can be achieved:

1. **Surveyed:** Accurately survey key points within the real world using the same (or different) technology to the AR system. The system is initialised with the coordinates of these positions. The registration error will be the sum of the error in the surveyed points and the error in the position measurement of the user.

2. **Manual:** The user manually adjusts the location of the virtual world until it coincides with the real one. The error depends worryingly on the quality of the user's alignment. (In fact, any AR system may require this capability to some extent. A VR headset may sit on a head slightly differently from one user to the next, so it is almost inevitable with today's non-invasive technology that some **fine** manual tuning is necessary.)

3. **Automatic:** Use a computer vision system to determine the relative position and orientation of the user in the environment. The error is singly sourced from that present in the vision system. The removal of the surveying stage is particularly clear in the case of video overlay AR where the image that is overlayed may itself be used to establish the direction and location of gaze of the user.

The advantages of the final option, in being almost totally automated, are overwhelming. Note that a vision-based AR system is **not** proposed for external use, but recommended for initial visual calibration outdoors. Targets **are** required, but only in a single place for "start-of-day" use. The user could be instructed to walk through a special doorway (pasted with appropriate targets) which would automatically invoke the registration. A visual system is also ideal to cope with position sensing technology that has any form of drift. A visual registration carried out every-so-often could provide a realignment to remove the drift. It is valuable, therefore, not to think of the vision system as an initialisation but as an infrequent yet periodic way of refreshing the registration. Even more generally, the vision system can be regarded as a slow-update absolute position device in a hybrid sensor scheme. Visually establishing registration also allows the main positioning technology to be relative rather than absolute.

5.2.3 Targets at Known or Unknown Locations?

Should the locations of the targets be known in advance? The positions of the targets, if known, are referred to as "survey data". When there are no survey data, the relative positions of the targets to one another may be established instead. Of course, the position of the camera is then only known in a relative sense and no absolute real-world alignment is achieved. However, if just three non-collinear targets of known location can be distinguished then the position of the camera is usually derivable absolutely (for exceptions see Haralick [57]). Needing to know the positions of three targets to establish registration is almost as constraining as having a full set of survey data. In fact, one

cannot do full registration, by its very definition, without some form of survey data. Dealing with the lack of survey data comes into its own in building virtual models from existing film sequences, or establishing relative camera positioning for the superimposition of computer graphics. With unknown target positions, it is possible to place virtual objects in a relative sense but only up to a global scale factor.

Sticking targets "willy-nilly" on a real-world scene does not help registration, because there is no way of knowing precisely where these targets have been put. If one places the targets on known feature points (and often this will not be possible) then one has effectively surveyed the targets because the positions of these features are known. A visual system for registration therefore essentially demands survey data, and though this is a disadvantage requiring extra effort, in practice targets are usually placed on regular grid structures on planar surfaces which greatly simplifies the process. Of course, target placement is always going to be subject to some error, and there are global error minimisation schemes to reduce the error in the positioning of the targets [56] [31]. Essentially, the systems recompute the actual position of the target rather than its nominal position and do achieve greater accuracy by this technique. To a certain extent there is a blurring of the distinction between improving the accuracy of inaccurate survey data thus, and establishing the positions of unsurveyed targets. However, the qualitative difference is the absence/presence of the tie in between the target sets and the real world coordinate system.

The recommendation is to survey targets as accurately as practicable, and if extra accuracy is needed in establishing camera orientation, then a global error minimisation scheme can be used to pinpoint the true target positions even more accurately. However, in a sparsely be-targeted environment such as Gosbecks a global scheme is less viable (the coverage map has to be continuous). In contrast, in a studio situation, with a densely targeted ceiling then global minimisation techniques come into their own [56].

5.2.4 Colour or Monochrome?

Should the computer vision algorithm work with colour images or monochrome images? As we have full control over target design and the task of target recognition is not the most complex in the world, it should be possible to work in the monochrome domain. The provides maximum compatibility and notwithstanding convenience: the development camera available is monochrome!

5.2.5 Single or Multiple Frame Operation?

Should the computer vision task work across a sequence of frames or on a per-frame basis? Complexity and accuracy have to weighed against simplicity and responsiveness respectively. In view of the well-defined nature of the task involved (finding camera location), it was felt that any benefits of temporal integration should be derivable at the positional level (whether in image or camera coordinates) and that frames should be processed individually. This allowed initial development to be based upon test images, with consequent reproducibility of behaviour and simplicity of approach. The less dependent on context a task is, the easier it is to debug.

5.2.6 Proposed System

The characteristics of the planned vision system are therefore: surveyed and distinguishable targets for automated operation; no global minimisation of target position error unless proven necessary. The actual processing falls into three stages: image processing of single gray-scale images to establish target IDs and position within image; establishing the transformation between real-world and image points; and turning this transformation into camera position and orientation. The literature usually only addresses the later two geometric stages of vision-based positioning systems and these are examined in the next chapter.

Although computer vision is traditionally regarded as computationally intensive and so suits a slow-update architecture, modern processors make vision processing at interactive rates entirely feasible. Therefore the slow-update quality of the proposed exterior vision system practically refers to the frequency at which targets are encountered rather than the latency of the image processing pipeline.

5.3 Image Processing

There is a vast amount of information in an individual image, and a sequence of images represents a considerable throughput of information. Computer vision may be described as the task of extracting from this super-abundance of data, facts pertinent to a particular application. It is not simply a matter of throwing away information in the image, but of “boiling down” the information content to distill the fractions of interest with as little loss as possible.

Due to the lack of literature on target detection and the wealth of image processing algorithms, the first step was to start almost totally *ab initio*: applying basic image processing operations to

target design elements to see what information could be cleanly extracted. Note, that the system developed is not to be considered as a research system. The principal aim of developing the system was to provide something that worked to effect other research ends. A clear gap was identified in the availability of public domain software in this area, and an intended by-product of the work was to plug that gap. To that end the software has subsequently been tested and rewritten to industrial standards, and an API has been designed, implemented and documented. Details of the API are provided in Appendix G. Despite being intended as a workman-like tool, some of the techniques invented during the development of the system may be considered novel.

The following sections discuss different target designs, and the two different approaches that were used to detect these targets. One approach was edge-based and the other was region-based. In the following discussions, the clarity of argument is maintained by regarding these as separate systems. However in reality, over the course of development the edge processing system took on a number of region based characteristics, until a totally region-based rewrite was effected. To avoid describing a continuum of systems, this artificial polarisation is introduced. The few aspects of the edge-based system which depended on region processing are pointed out - these do not undermine the separation. The edge processing is described before the target designs, because it is the foundation stone for the first generation of targets. The region processing is described after the target designs, because the final nature of the region processing is based on the nature of the final target design.

5.4 Edge Processing

One of the most fundamental operations in image processing is edge detection. It allows 2D raster-maps to be turned into geometric curves, from whose properties deductions about the scene can be made. The steps can be described as follows:

Filtering: The difference between adjacent pixel values is an indication of local intensity change. However, differencing methods are particularly susceptible to noise, so it is typical to smooth an image first to reduce the sensitivity of the overall edge detector to noise.

Edge Enhancement: This emphasises regions of the smoothed image where there are significant changes in local intensity: these are usually found by looking at the magnitude of the gradient field.

Edge Detection: Some further criteria are used to determine which of the pixels of large magnitude gradient values are likely to be edge points.

Localisation: By matching an appropriate geometric primitive with sets of edge points, it is possible to identify both the nature of the curve appearing in the scene and to determine its characteristics. Obviously this part of edge detection depends on the type of target used, and information on this phase of the processing largely appears in the sections on target design.

The following discussion documents the particular edge detection procedure used, which is based on the Canny edge detector. Most information is provided on the algorithms that were specially adapted to meet the needs of the application, though the full edge detection process is given.

5.4.1 Filtering

An edge point marks the location of a significant local intensity change in the image. The requisite gradient calculation, if based on just the intensity values of two adjacent pixels, will be subject to noise and errors associated with discrete computation. Consequently filtering is commonly used, though at the natural expense of edge strength. In a Canny edge detector the image I, is convolved with a Gaussian filter G, to form a smoothed resultant image S.

$$S[i,j] = G[i,j,\sigma] * I[i,j]$$

where i,j are pixel coordinates, and σ represents the spread. The Gaussian filter used is:

$$\begin{aligned} G[i,j] &= e^{\frac{-(i^2+j^2)}{2\sigma^2}} \\ G[i,j] * I[i,j] &= \sum_{k=1}^{m}\sum_{l=1}^{n} G[k,l]\, I[i-k, j-l] \\ &= \sum_{k=1}^{m}\sum_{l=1}^{n} e^{\frac{-(i^2+j^2)}{2\sigma^2}}\, I[i-k, j-l] \\ &= \sum_{k=1}^{m} e^{\frac{-(k^2)}{2\sigma^2}} \left(\sum_{l=1}^{n} e^{\frac{-(l^2)}{2\sigma^2}}\, I[i-k, j-l] \right) \end{aligned}$$

The term in the brackets is the convolution of I with a 1D Gaussian function, and the term outside the brackets represents a further convolution against an orthogonal 1D Gaussian function. The 2D Gaussian convolution can thus be performed as two successive 1D Gaussian convolutions. The

Gaussian function is said to be "separable". This is beneficial, as the computational complexity of a 1D filter grows linearly with its width whereas the computational complexity of a 2D filter grows quadratically with its area.

Filter Size

One feature of the software is that the size of the Gaussian kernel employed is not fixed but is dynamically computed, based both on the required level of accuracy of the filtered image and the value of σ, the single parameter which determines the spread of the Gaussian filter. σ may be supplied as a run time parameter. For a given σ how big should the filter be? A filter is defined as a matrix of values:

$$F(i,j) \quad \text{where} \quad -n \leq i \leq n, -n \leq j \leq n, \quad n \in \mathbb{N}, i,j \in \mathbb{Z}$$

We want n to be large enough to approach the full value of the convolution with the infinite Gaussian, within the accuracy of the image format (*i.e.* number of gray levels). We solve for n, with a given σ and $levels$ intensities.

$$\begin{aligned}
\frac{\int_{-n}^{n}\int_{-n}^{n} e^{\frac{-(x^2+y^2)}{2\sigma^2}}\,dy\,dx}{\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{\frac{-(x^2+y^2)}{2\sigma^2}}\,dy\,dx} &= 1 - \frac{1}{2 \times levels} \\
\Rightarrow Erf^2(\frac{n}{\sqrt{2}\sigma}) &= 1 - \frac{1}{2 \times levels} \\
\Rightarrow n &= \sqrt{2}\sigma Erf^{-1}\left(\sqrt{1 - \frac{1}{2 \times levels}}\right) \\
\Rightarrow n &= k(levels) \times \sigma \\
\text{where } k(levels) &= \sqrt{2} Erf^{-1}\left(\sqrt{1 - \frac{1}{2 \times levels}}\right)
\end{aligned}$$

n is therefore linearly proportional to σ. Software was written to calculate the constant of proportionality: $k(levels)$ given $levels$. Fortunately, the mathematical library for the 'C' language under UNIX supplies Erf and so the required inverse function can also be readily written, as Erf is strictly monotonic (see Figure 5.1). The algorithm to find the inverse of a monotonic function can be compactly expressed in ML as in Appendix E. Table 5.1 shows the values of function k given a number of different gray-scale levels. k is not very sensitive to levels, and $k = 3$ is sufficient for our purposes (giving 0.2% accuracy for 256 gray levels). For a modest Gaussian $\sigma = 1$, this supports the fact that

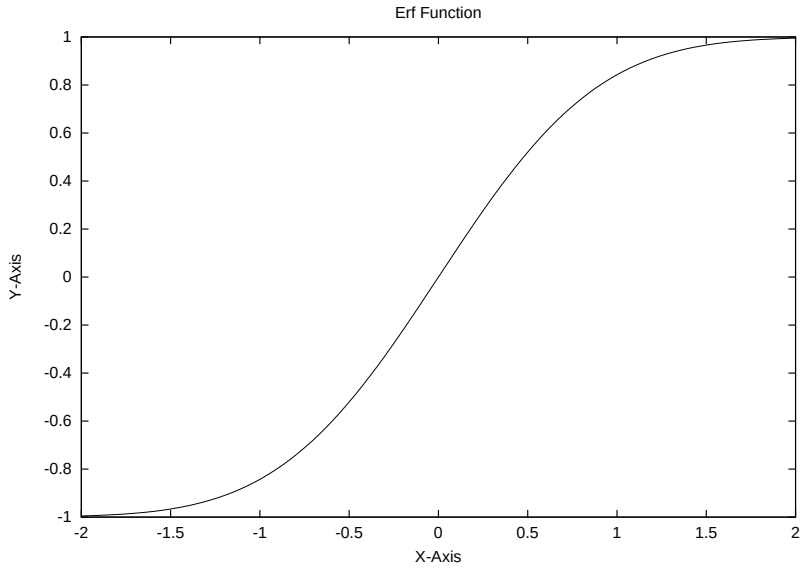

Figure 5.1: Erf function

gray levels	bits per pixel	k
4	2	1.8
16	4	2.4
256	8	3.3
65536	16	4.6
16777216	24	5.7

Table 5.1: Values of $k(levels)$

the small Gaussian filters in the textbooks are 7×7 *i.e.* $-k\sigma$ to $k\sigma)$ in each dimension.

5.4.2 Edge Enhancement

The first step in edge enhancement is to compute the gradient field from the smoothed image intensity map S. We examine different ways of calculating the gradient and present these in matrix form. The partial derivatives of S can be computed by subtracting intensities of horizontally and vertically adjacent pixels:

$$S_x \left[i, j + \frac{1}{2} \right] \quad = \quad S[i, j+1] - S[i, j]$$

$$= \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$\begin{aligned} S_y\left[i+\frac{1}{2}, j\right] &= S[i+1,j] - S[i,j] \\ &= \begin{bmatrix} -1 & 1 \end{bmatrix} \end{aligned}$$

Note that the partial derivatives have not been calculated at a common point; to do so we can interpolate S_x and S_y at $\left[i+\frac{1}{2}, j+\frac{1}{2}\right]$

$$\begin{aligned} S_x\left[i+\frac{1}{2}, j+\frac{1}{2}\right] &\approx \frac{S_x[i, j+\frac{1}{2}] + S_x[i+1, j+\frac{1}{2}]}{2} \\ &= \frac{S[i,j+1] - S[i,j]) + S[i+1,j+1] - S[i+1,j]}{2} \\ &= \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix} \end{aligned}$$

$$\begin{aligned} S_y\left[i+\frac{1}{2}, j+\frac{1}{2}\right] &\approx \frac{S_y[i+\frac{1}{2}, j] + S_y[i+\frac{1}{2}, j+1]}{2} \\ &= \frac{S[i+1,j] - S[i,j]) + S[i+1,j+1] - S[i,j+1]}{2} \\ &= \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} \end{bmatrix} \end{aligned}$$

To further move this common point to $[i, j]$ we can use a 3×3 operator:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad S_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Notice, however, that the pixel values being used are not equidistant from the point in question, so weightings should be applied to closer points. One solution is:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

This is known as the Sobel operator. However, the shifting of the partial derivative images by half-a-pixel is hardly a crime for it occurs globally across the image, and for our purposes may just be regarded as a camera offset that is taken into account. Indeed the Canny edge detection algorithm

uses the 2×2 filters given above. The magnitude and orientation of the gradient $(P[i,j], Q[i,j])$ may be computed by:

$$\begin{aligned} M[i,j] &= \sqrt{P[i,j]^2 + Q[i,j]^2} \\ \theta[i,j] &= \tan^{-1}(P[i,j], Q[i,j)) \end{aligned}$$

$tan^{-1}(x,y)$ is used as unlike $arctan$, its image is the full angular range $-\pi < tan^{-1}(x,y) \leq +\pi$. A call to the corresponding complicated floating point operation (namely $atan2$) for every pixel is going to be expensive. How big would a 2D look-up table have to be to avoid this computation? For a 256 level intensity image:

$$\begin{aligned} -510 < \quad S_x \quad < 510 \\ -510 < \quad S_y \quad < 510 \end{aligned}$$

The number of table entries required would be $(510 \times 2 + 1)^2 = 1042441$. Using a floating-point representation of 8 bytes the storage required would be rather high namely 8 Mbytes. While computers do have this memory available, the initialisation time for this array will not be inconsiderable The compromise solution is to store only the members of the look-up table that are most likely to be accessed. A 256×256 sub-array in the centre of the table was chosen.

```
#define TABLE_SIZE 256
#define TABLE_MID (TABLE_SIZE / 2)
#define TABLE_LOWER ( - TABLE_MID )
#define TABLE_UPPER ( TABLE_MID  - 1  )
static int orientation_table[TABLE_SIZE][TABLE_SIZE];
/* returns orientation of vector (P,Q) in degrees */
int get_orientation(int P, int Q)
{
       if
       (      ( Q < TABLE_LOWER) || ( Q > TABLE_UPPER)
              ||
              ( P < TABLE_LOWER) || ( P > TABLE_UPPER)
       )
       {
              return (int) (180.0 + (atan2( (double) Q, (double) P) / PI) * 180.0);
       }
       else
       {
              return orientation_table[TABLE_MID+Q][TABLE_MID+P];
       }
}
```

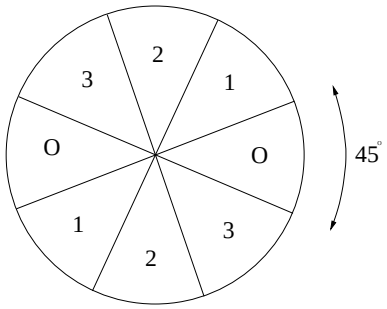

Figure 5.2: Sectors for gradient quantisation

In practice, the table size was chosen to be sufficiently large so that the first branch of the code was never taken. The storage and initialisation tasks were thus reduced by a factor of 16.

5.4.3 Edge Detection

The edge detection in the Canny method consists of two steps: non-maxima suppression and then thresholding.

Non-Maxima Suppression

An edge in the image M may be represented by a thick ridge where the gradient value is high. To detect the edges, ridges in M must be thinned to leave only those pixels where there is the greatest rate of change in the original image — a process called non-maxima suppression. The gradient orientation angle may be quantised to a sector number within the range $0 \cdots 3$ as represented in Figure 5.2.

$$\phi[i,j] = Sector(\theta[i,j])$$

Each element of M is compared to the two adjacent pixels along the quantised line of gradient. If it is not greater than both adjacent pixels, then it is set to zero. Let the function nms represent this process. Thus the non-maximally suppressed image N is:

$$N[i,j] = nms(M[i,j), \phi[i,j])$$

Note that non-maxima suppression must not be performed on the array M itself *i.e.* elements should not be cleared to zero as they will affect the detection of other local maxima, depending on the direction of scan through the image.

Suppression Algorithms

It was noticed that edges resulting from targets of concentric black and white rings were very clean. However, the edges were not connected, and circles were broken precisely at the 45° angle positions, when the sector number is a little ambiguous. This anisotropy implied the algorithm was incorrect. As it was planned to use the connectivity of these circular edges for target detection, an investigation was undertaken. There are four boolean values $b_0[i],\ 0 \leq i \leq 3$, describing whether there is a local maximum in a sector direction:

$$\begin{aligned}
b_0[0] &= (I[x,y] > I[x+1,y]) \cap (I[x,y] > I[x-1,y]) \\
b_0[1] &= (I[x,y] > I[x+1,y+1]) \cap (I[x,y] > I[x-1,y-1]) \\
b_0[2] &= (I[x,y] > I[x,y+1]) \cap (I[x,y] > I[x,y-1]) \\
b_0[3] &= (I[x,y] > I[x+1,y-1]) \cap (I[x,y] > I[x-1,y+1])
\end{aligned}$$

Canny just chooses one of these values, resulting in breaks in the edge. Let us call Canny's method Solution 0.

Solution 1

Define a new boolean condition which includes adjacent sectors too:

$$b_1[i] = \bigcup_{\substack{j=0 \\ j \neq (i+2) \bmod 4}}^{3} b_0[j] \qquad 0 \leq i \leq 3$$

This gives a connected edge, but one which is overly so *i.e.* not maximally thinned! The $\bigcup$ implies a boolean (*cf.* arithmetic) add of the component conditionals.

Solution 2

Redefine sectors with a displacement of 22.5°, to define a new boolean:

$$b_2[i] = b_0[i] \cup b_0[(i+1) \bmod 4] \qquad 0 \leq i \leq 7$$

Here we look at the two nearest old fashioned sectors. This gives a connected edge, but not sufficiently less connected than Solution 1.

Solution 3

Redefine sectors to be eight in number, so angle is clearly within an old sector or between two old sectors.

$$\begin{aligned}
b_3[0] &= b_0[0] \\
b_3[1] &= b_0[0] \cup b_b[1] \\
b_3[2] &= b_0[1] \\
b_3[3] &= b_0[1] \cup b_b[2] \\
b_3[4] &= b_0[2] \\
b_3[5] &= b_0[2] \cup b_b[3] \\
b_3[6] &= b_0[3] \\
b_3[7] &= b_0[3] \cup b_b[0]
\end{aligned}$$

In fact, despite being intuitively promising this gives breaks in the edge. Solution 2 is clearly superior to Solutions 0, 1, and 3. Proceedings so far have been on an *ad hoc* basis, and it is time to work things out properly.

"Correct" Solution 4

The component of $b_0[i]$ with sector angle $s[i]$ (0°,45°,90° or 135°) perpendicular to the edge at angle α (the direction of the gradient field, $0° \leq \alpha < 360°$) is $|\cos(s[i] - \alpha)|$. The modulus operator is used as the edge is not directed. Also as the function repeats every 180° the case $\alpha > 180°$, *etc.* need not be treated separately. The probability that there is a maximum, is given by:

$$\sum_{i=0}^{3} b_0[i] \, |\cos(s[i] - \alpha)|$$

Note the analogy between this and the superposition of quantum states as $b_0[i]$ is either a 0 or a 1, and the coefficient is a probability. We need a threshold k to define the maximal edge *i.e.* where

$$\sum_{i=0}^{3} b_0[i] \left|\cos(s[i] - \alpha)\right| > k$$

Let us work out a suitable value of k. Say the Canny test fails for the actual sector involved, but works for one of the adjacent ones. This condition corresponds to where Canny lets us down *i.e.* $b_0[i]$ is zero, but $b_0[(i+j) \bmod 4]$ is 1 where $j = -1$ or 1. The minimum value of $|cos(s[(i+j) \bmod 4] - \alpha)|$ is $\cos(45^\circ)$ *i.e.* $\frac{1}{\sqrt{2}}$. This corresponds to the case where $|\cos(s[i])|$ is 1. Therefore we want a threshold just below $\frac{1}{\sqrt{2}}$ say 0.707 and through experiment this is shown to work well.

$k = 0.5$	was found experimentally to be overly connected
$k = 1.0$	was found experimentally to be underly connected

In fact $k = 0.707$ connects all the edges correctly, with very few spurious pixels. Note that Canny gives a minimally thin king-wise[1] connected edge. This new (but more correct algorithm) gives a slightly richer connection.

Thresholding

To remove false edges caused by noise and weak image feature, all pixels in $N[i,j]$ below a certain threshold are set to zero. Choosing a threshold is fraught with problems, so a double thresholding algorithm is used which makes the choice of threshold less critical and salvages important but weak features. N is thresholded to images T_1 and T_2 where the second threshold is roughly twice that of the first. The contouring algorithm builds up 8-connected edges using data from T_2; where 8 connection is lost it is sought in T_1 until a connection is made with an edge in T_2. In this way, weaker sections of a significant edge are effectively bridged over. The recursive algorithm is shown in Figure 5.3. Note that the input image has been thresholded to three levels 0, 64 and 255 representing the intervals $N[i,j] < T_1$, $T_1 < N[i,j] < T_2$ and $N[i,j] > T_2$ after non-maximal suppression. "subtrace" is the essential recursive function, while "trace" copes with the possible dual thresholded nature of the edge discovered.

[1]Two pixels horizontally, vertically or diagonally adjacent are said to be *king-wise* connected or 8-connected *cf. rook-wise* or 4-connectivity. The more poetic nomenclature is based on the permitted moves of chess pieces.

```
#define REPLACE_COLOUR 128
int count;                                  // accumulating number of pixels in current edge
int maxdepth;

/*
        subtrace & trace:

                        mutually recursive functions for elucidating a single edge

        arguments:

        I               - four-level input/output image

                                0       intensity value less than threshold 1
                                64      intensity value between thresholds 1 and 2
                                255     intensity value above threshold 2
                                128     pixel has been assigned to an edge

        C               - output image containing edges each having a unique integer ID

        (x,y)           - pixel coordinate
        depth           - recursion depth
        edge_colour     - colour or ID of current edge
        follow_colour   - current threshold value used for growing edge either 64 or 255
*/

void subtrace(Image I, Image C, int x, int y, int depth, int edge_colour, int follow_colour)
{
        if ( I[x,y] == follow_colour )      // is pixel above appropriate threshold ?
        {
                C[x,y] = edge_colour        // keep separate record of growing edges
                I[x,y] = REPLACE_COLOUR;    // tick off as edge pixel
                count++;

                if ( depth > maxdepth )
                {
                        maxdepth = depth;
                }

                for(j=-1;j<=1;++j)
                {
                        for(i=-1;i<=1;++i)
                        {
                                if ( Neighbour(x,y,x+i,y+j) )
                                {
                                        // recursive step - trace though 8-connected neighbours
                                        trace(I, C, x + i, y + j, depth + 1, edge_colour);
                                }
                        }
                }
        }
}

void trace(Image I, Image C, int x, int y, int depth, int edge_colour)
{
        subtrace(I, C, x, y, depth, edge_colour, 255);

        if ( maxdepth == depth )            // no connections of 255 found
        {                                   // so connect through lower threshold
                subtrace(I, C, x, y, depth, edge_colour, 64);
        }

}

/*
        find_edge: top level function to find all edges
                   scans through image pixels looking for places to start
                   edge following
 */

Image find_edge(Image I_arg)
{
        Image C;
        Image I         = I_arg; // copy argument as algorithm writes to Image I
        int edge_colour = 1;     // colour or ID of first edge

        maxdepth        = 0;
        count           = 0;

        clear_image(C);         // zero all pixel values

        for(int y=0;y < image_height(I) ;++y)
        {
                for(int x=0;x < image_width(I) ;++x)
                {
                        // edge trace starts through high threshold
                        subtrace( I, C, x, y, 0, edge_colour, 255 );
                        if ( count > 0 )
                        {
                                // found a non null edge - prepare for new edge
                                edge_colour = edge_colour + 1;
                                maxdepth    = 0;
                                count       = 0;
                        }
                }
        }
        return C;
}
```

Figure 5.3: Double-thresholding edge-following algorithm

Edge Detection Conclusions

It is a considerable surprise the textbook [58] Canny algorithm does not work The discovery was only chanced upon as the fiducials images were so clear, that there was no way that an edge detector worth its salt would have introduced a break in the edge. The textbook stuffs the non-maxima-suppressed magnitude back into the magnitude array as the algorithm is processing. This was avoided as "highly dangerous", but was eventually tried when Canny failed just in case it was a critical part of the algorithm. However, on using the same array, edge "echos" were erroneously detected revealing a second error in the textbook presentation.

5.4.4 Localisation

How can a series of edges be recognised as a target? Note that the following arguments are very general, and could equally apply to areas. It is possible to collect a number of statics on an individual curve, such as the centroid:

$$(\bar{x}, \bar{y}) = \left(\frac{\sum_{i=1}^{n} \sum_{j=1}^{m} iB[i,j]}{\sum_{i=1}^{n} \sum_{j=1}^{m} B[i,j]}, \quad \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} jB[i,j]}{\sum_{i=1}^{n} \sum_{j=1}^{m} B[i,j]} \right)$$

The orientation of a curve (which is defined when the curve is elongated) is defined as the axis of second moment *i.e.* the line for which the sum of the squared distances between object points and the line is minimised. The expression to minimise is:

$$\sum_{i=1}^{n} \sum_{j=1}^{m} r_{ij}^2 B[i,j]$$

where r_{ij} is the perpendicular distance from the line to point $[i,j]$ on the curve. Let us assume that the equation of the axis of second moment is $x\cos\theta + y\sin\theta + c = 0$ where θ is the angle the line makes with the x-axis and c is the distance of the line from the origin. Now the distance of point (x,y) to this line is $x\cos\theta + y\sin\theta + c$, So we have to minimise:

$$\chi^2 = \sum_{i=1}^{n} \sum_{j=1}^{m} (x\cos\theta + y\sin\theta + c)^2 B[i,j] \tag{5.1}$$

Differentiating with respect to c, and setting $\frac{\partial(\chi^2)}{\partial c}$ to zero, gives:

$$c = -(\bar{x}\cos\theta + \bar{y}\sin\theta)$$

Substituting this back into 5.1, and changing variables to $x' = x - \bar{x}$ and $y' = y - \bar{y}$ gives:

$$\chi^2 = a\cos^2\theta + b\sin\theta + c\sin^2\theta \tag{5.2}$$

where:

$$a = \sum_{i=1}^{n}\sum_{j=1}^{m}(x'_{ij})^2 B[i,j] \qquad b = 2\sum_{i=1}^{n}\sum_{j=1}^{m}x'_{ij}y'_{ij}B[i,j] \qquad c = \sum_{i=1}^{n}\sum_{j=1}^{m}(y'_{ij})^2 B[i,j]$$

Using double angle formulae we can express 5.2 as:

$$\chi^2 = \frac{1}{2}(a+c) + \frac{1}{2}(a-c)\cos 2\theta + \frac{1}{2}b\sin 2\theta$$

Differentiating with respect to θ, and setting $\frac{\partial(\chi^2)}{\partial\theta}$ to zero, gives:

$$\tan 2\theta = \frac{b}{a-c}$$

which provides us with a value for θ the angle of orientation of the axis of elongation. Note that the curve will not have a unique axis of orientation if and only if $a \approx c$ and $b \approx 0$. This will be the case when the target is a circle that is square on to the camera. However, ignoring perspective distortion, a circle at angle will appear as an ellipse which will have a defined angle of orientation. Calculating this angle of orientation will help us to effectively "rectify" the circle.

5.5 Particular Targets

Now that the ability to detect edges has been developed, the question arises of what configuration of edges should be incorporated into a target of good design. The following sections introduce the various attempts at target designs. With hindsight, it is possible to identify the necessary characteristics of a good target:

Detectability: A target must have a property that makes it stand out clearly from the rest of the scene.

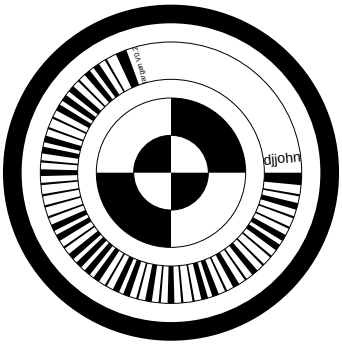

Figure 5.4: Initial target design

Distinguishability: It must be possible to tell each target apart from any other.

Diversity: How many different targets can be produced? If there are only a small number of different targets then the area that can be covered may be limited.

Locatability: Some means must be provided of locating a particular point of the target within the scene, so exact coordinates can be established for subsequent accuracy of processing.

5.5.1 Circular Barcode

The initial target design can be seen in 5.4. It was clear from sample images that the feature-size of the circular barcode was too small. It would be impossible to ever extract the encoded information (an ASCII string) unless the target was impractically close to the camera. Secondly, the components of this design are not well integrated *e.g.* finding the surrounding black circular band still leaves the problem of locating the barcode.

5.5.2 Spiral

Inspired by the highlighted problem of the variable camera to target distance the next target design was a logarithmic spiral which almost uniquely does not change shape when scaled. A spiral also has the property of windiness (of turning around and around a point many times) which few objects in a scene will possess, more formally identifiable as the mathematical "winding number" η which is a complex integral calculated with respect to a point z_0, and a curve C:

$$\eta(C, z_0) = \oint_C \frac{dz}{(z - z_0)}$$

The polar equation of a logarithmic spiral is:-

$$r = Ae^{B\theta} \quad \text{where} \quad A, B \in \mathbb{R} \quad \text{are constants}$$

A just controls the scale, but it is possible to determine an optimum value of B (the "windiness") for detection. To use the image area effectively even towards the centre of the spiral when the turns get tighter, two adjacent resolvable turns should roughly be within the resolvable distance d of the centre. These conditions are:

$$d = Ae^{B(\theta+2\pi)} - Ae^{B\theta} \quad \text{and} \quad d \approx Ae^{B\theta}$$

Solving for B strangely yields a value that also occurs in the natural kingdom (*e.g.* mollusc shells):

$$B = \frac{log2}{2\pi} \approx 0.11$$

Refinements of the original spiral target (Figure 5.5) to yield a pure spiral curve under edge detection brought the terminating turn within the confines of the paper (Figure 5.6); blended this turn into the background (Figure 5.7); and then moved to a single rather than double edge by grey-scaling. Figures 5.8, 5.9 and 5.10 represent the final spiral target design, with low, medium and high windiness respectively. Even though the gray-scale spiral produces a single edge, if the edge threshold was low enough a faint secondary spiral edge corresponding to the mid-point of the gray-scaling was revealed. This is a salutary lesson in the nastiness of thresholds: by an inappropriate choice the double spiral problem could be re-introduced!

Tests were undertaken using spiral fiducials with various windiness factors (B values of 0.05, 0.1, 0.2 and 0.4) at various effective distances from the camera (by changing the zoom setting z), see images in Figures 5.11, 5.12, 5.13 and 5.14. The relationship between the best B and z is not clear. There may be a single optimum B, or a slowly varying optimum B for the typical values of z in use. Though the results were noisy, an experimentally optimal value of B was found to be in the region of 0.11 as theoretically predicted.

Obtaining the highest winding number is dependent on reasonably accurately locating the centre of the spiral, and appropriate points to start and finish the integral. A number of effective algorithms were compared to locate these points. It was found difficult to obtain a characteristically high winding number to identify a spiral with typical values of η being in the range 1 to 2. So although spirals were

Figure 5.5: Infinite spiral

Figure 5.6: Finite spiral

Figure 5.7: Angle-blended spiral

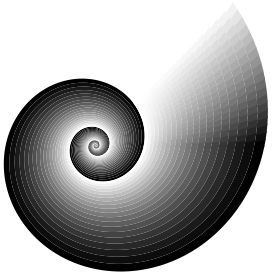

Figure 5.8: Loose spiral

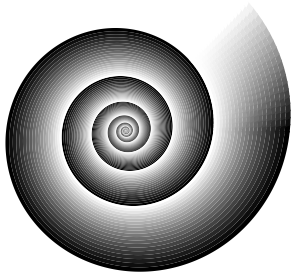

Figure 5.9: Intermediate spiral

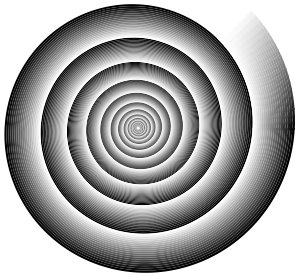

Figure 5.10: Tight spiral

Test	Relationship/Property	Domain	Recommended Use
1	one inside another	equivalence relation	selection
2 (slack)	shared centroids	equivalence relation	selection
2 (tight)	shared centroids	equivalence set	verification
3	same aspect ratio	equivalence relation	selection
4	two outer curves in ratio 8:9	equivalence set	verification
5	outer curve is black $\rightarrow$ white	equivalence set	correction

Table 5.2: Best experimentally established uses for circular target tests

successfully identified, η was not much higher than for a circle in general.

The highest η obtained was around 7.2 when the target was near the camera: Figure 5.15 shows how the winding number varies across the target area of the image shown in Figure 5.14. In this case, the sharp peak in η very successfully identifies the spiral and there is a fascinating resemblance to the ziggurat-shape of the spiral minaret of the Great Mosque at Sarama in Iraq built in 850 AD (Figure 5.17).

Detail of the summit as shown in Figure 5.16 reveals that it is surprisingly flat and thus cannot be used to find the centre of the spiral more accurately: there had been some hope that perhaps the winding number could be used to find the spiral's centre to sub-pixel resolution. On the other hand, the flat summit indicates that it is not necessary to find the pixel's centre very accurately to obtain a maximum winding number. The conclusion is that the technique is not limited by the accuracy of the computed centre of the spiral, but by its intrinsic nature. No way was determined to give each spiral target a different identifier.

5.5.3 Circular Targets

Given that spirals were not much better than circles led to the circular targets' design in Figure 5.18. Numbering the 8 concentric rings from the inside out: ring 8 is always black; ring 7 is always white; while rings 1 to 6 encode the target ID.

Detecting Circular Targets

Circular edges within a scene form target equivalence classes. The sequence of operations in target detection is: creation of equivalence sets using selection tests; potential correction of equivalence sets (to increase true positives); and verification of equivalence sets as targets (to decrease false negatives).

Figure 5.11: Distant spiral targets

Figure 5.12: Mid-range spiral targets

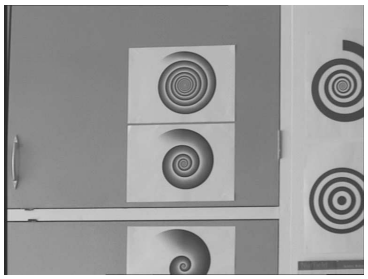

Figure 5.13: Close spiral targets

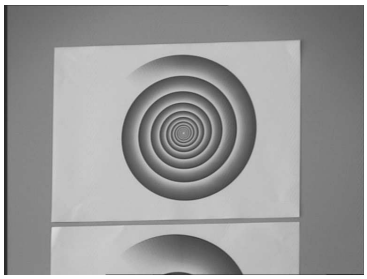

Figure 5.14: Very close spiral target

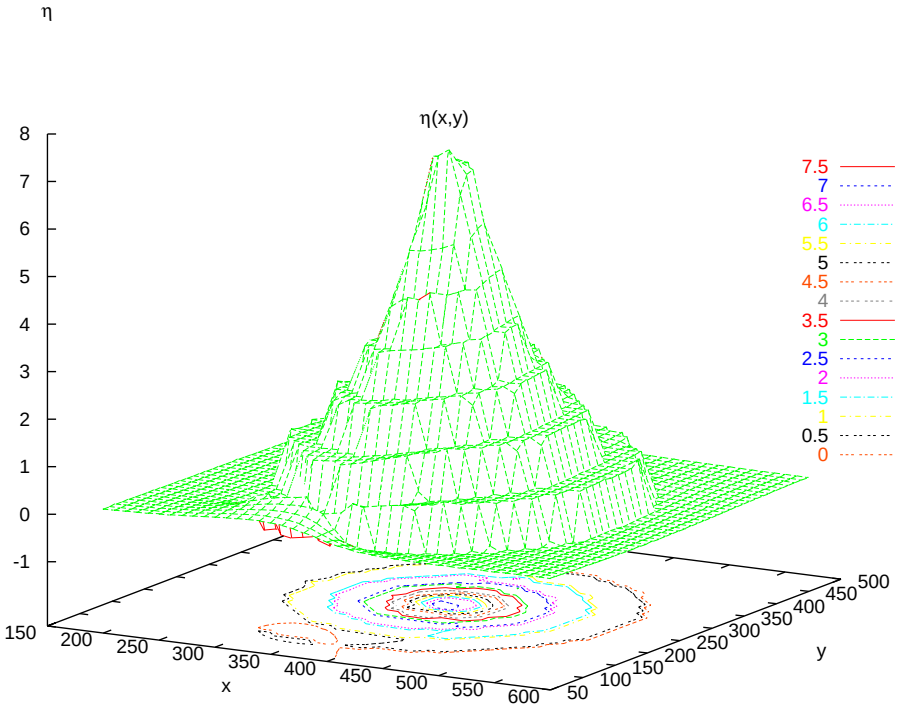

Figure 5.15: Winding number as a function of position over spiral target image

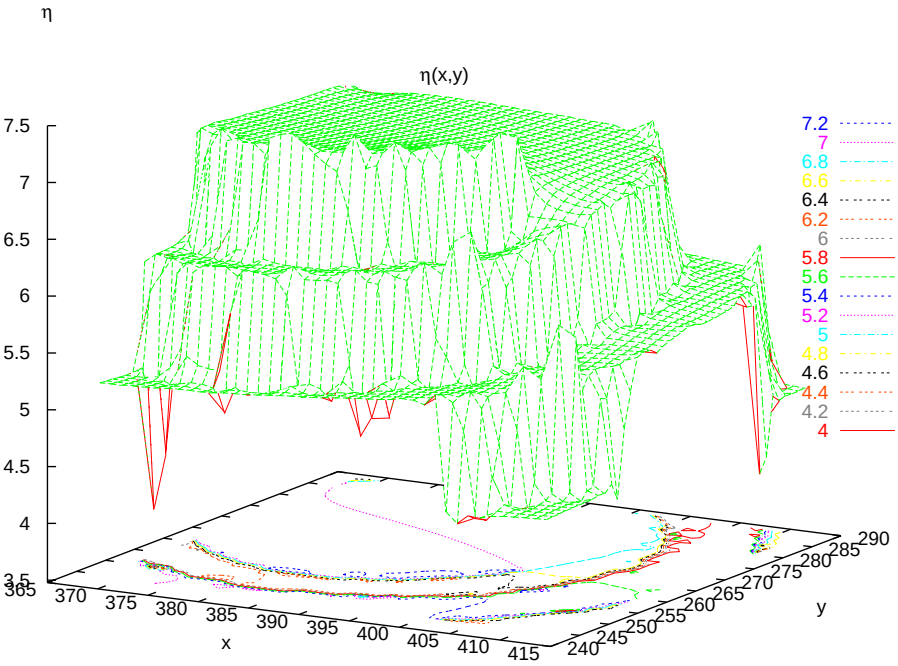

Figure 5.16: Winding number as a function of image position: summit detail of Figure 5.15

Figure 5.17: Spiral ziggurat of Samara

The inter-curve relationships and intra-target properties used are given in Table 5.2. Test tolerances were determined experimentally through trial and error. This is laborious and feels unsatisfactory, but is the only way to use each test as effectively as possible in the field. One test (shared centroids) is used to both group curves and verify targets, because given the same tolerance, all the curves sharing the same centroid is a stricter condition and more relevant to a target, than simply pairs of curves doing so. In practice a spurious outer curve was sometimes detected (*e.g.* the edge of the paper could act in this way) destroying a perfectly valid target. Experiments showed that peeling away an outer curve representing the wrong intensity transition always assisted and never hampered detection.

Distinguishing Circular Targets

How can the identifiers be deciphered? Some of the techniques used are based on the curve data already extracted and some are based on re-processing the regions where targets were found.

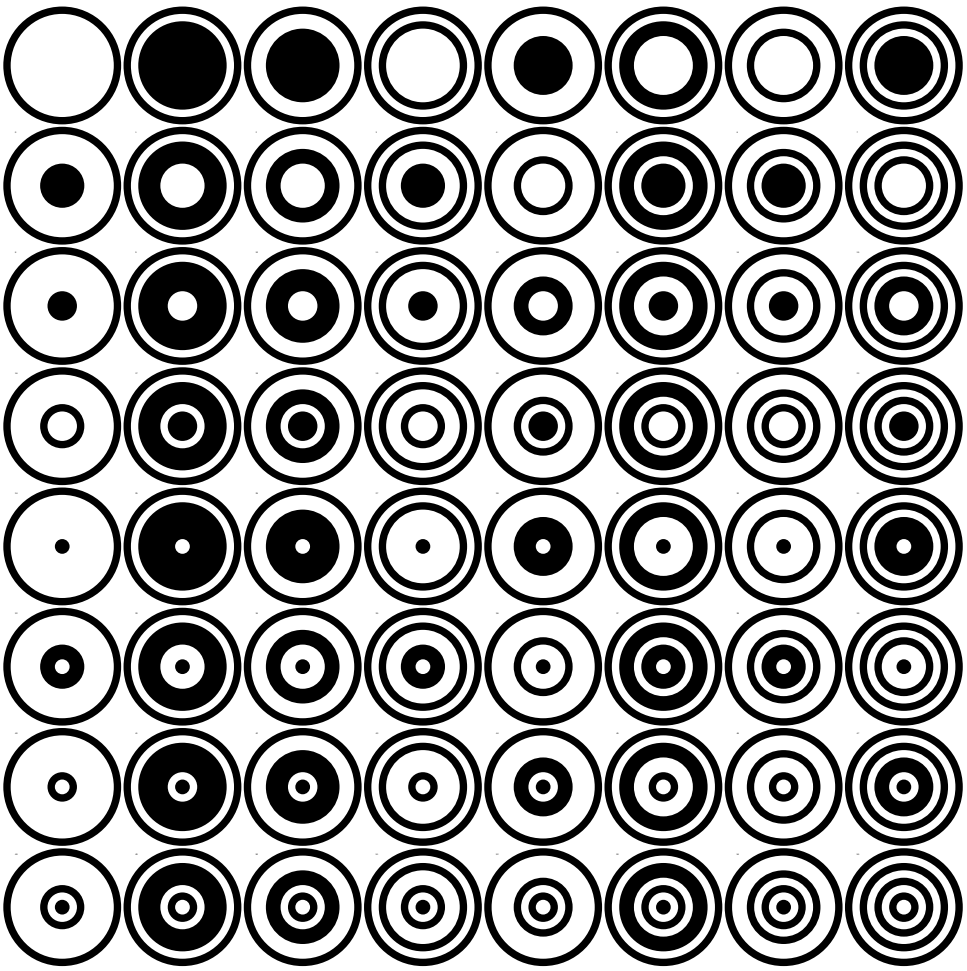

Figure 5.18: Circular targets

Process Target by Radial Histogramming

Any elliptical distortion of the target (detected as second moment elongation) is corrected for, equivalent undistorted bitmaps are shown in Figure 6.5. The radial intensity distribution of each target detected is then histogrammed. Also histogrammed radially were the gradient field; the edge data (edge pixels add one to a bin); and the region data ("black" pixels subtract one from a bin and "white" pixels add one to a bin). The latter case removes the problem of determining an edge threshold to detect peaks or an intensity threshold to detect steps, as zero crossings give the edges; otherwise the histogram has also to be corrected by the factor $\frac{1}{r}$ (where r is the radial distance) because fewer pixels constitute the inner rings. Let the identifier produced be called ID_{hist}.

Process Target by Slice

The minimum bounding rectangle oriented at the angle of elongation for each target is computed. Determining a target's identity is reduced to examining a linear sequence of intensity values along the notional line bisecting this rectangle parallel to the long edges. This is the now the same as the previous histogram problem. In fact, two sets of samples are taken: one along the line proceeding in one direction from the centre of the target and the other in the opposite direction. Two ID values are produced ID_{right} and ID_{left}. The names are just for reference as the line may not be horizontal.

Process Target by Size

The size of the bounding box of each component curve relative to to the size of the bounding box of the outermost curve gives an indication of the target identifier; let the value obtained be called ID_{size}.

Process Target by Population

The number of pixels in each of the curves gives an indication of a target's ID. The number of pixels in ring i is proportional to $C = 2\pi i$. A global scale factor is calculated from the total number of pixels in all the detected curves and for each i, an expected number of pixels is computed. By looking for the closest match, we can guess i for each detected curve. The target identifier produced is called ID_{pop}.

Results

Contrary to expectation as it uses an discrete integrated property, ID_{pop} is least often correct about the ID of a target. Perhaps the number of pixels in a elliptical curve does not scale linearly with the size of the curve given the modified Canny algorithm? In this light a region-based ID_{pop} may yet produce the most reliable results but this was not investigated. ID_{size} is the most reliable measure; ID_{hist} is also good followed by ID_{right} and ID_{left}. It is difficult to know how to combine differing ID determinations for the best combination of reliability and correctness. In practice, the following test for a valid ID was around the most successful.

$$valid\_id = (ID_{pop} == ID_{size}) \cup (ID_{left} == ID_{size}) \cup (ID_{right} == ID_{size})$$

When the target is large in the visual field all ID determinations will agree. For example, the targets close to the camera in Figure 5.19 are unambiguously identified in the edge processed image

Figure 5.19: Circular targets close to camera

Figure 5.20: Edge processed

Figure 5.21: Test image: mixed range targets

Figure 5.20. Disagreements start when the target is below a certain size, and where the target is too small clearly no correct ID determination can take place. Individual ID determination algorithms can themselves flag a failure if there is insufficient consistency in the data.

Overall Findings

Testing of target detection and identification largely took place using fixed test images such as Figure 5.21. Results were dependent on choosing good upper and lower thresholds values and for optimality these thresholds had to be varied according to the scene being processed. There was unease about aspects of the method, because though the eye could clearly see a target in a dimmer region of the scene, this target would not be detected simply because the edges were weaker. Some kind of local normalisation appeared to be called for, which suggested region based processing. Despite all the verification tests described above, the occasional false positive target would emerge. What is peculiarly special about the circular targets, that we can use to stop this happening? Each ring is only surrounded by one other ring, and representing a target as a Region Adjacency Graph [58] is

particularly illuminating.

A Region Adjacency Graph (or RAG) represents a region decomposition of a scene as a graph, where each region is a node and each arc represents the adjacency relationship between regions. Each node may in addition be labelled by the colour of the corresponding region. Many different RAGs can be derived from an image depending on the nature of the region decomposition. A RAG itself is trivial to produce, though the earlier region decomposition may naturally be of arbitrary complexity. RAGs have found uses in image interpretation applications.

In a RAG representation a circular target is a characteristic linear feature, consisting of a chain of black and white nodes. A further filter was thus placed on detected targets. This additional condition was that they had to be a linear feature in RAG space. All at once, the false positive rate dropped to zero. The details of the processing are left to the next section, suffice to say that the result was so impressive that immediate thoughts of a region-only processing system came to mind.

The edge-based system worked well, particularly with relatively even illumination and where targets were fairly large in the scene. Even where targets were misidentified because they were small they were still found, and other techniques were eventually used to successfully guess the IDs of these misidentified targets (see Section 6.6). In short, the system showed graceful degradation with distance, correctly finding objects as targets even where it was clearly impossible to extract their IDs. The major reservations about the system were: the complexity of many of the tests; the number of runtime parameters that had to be set correctly for reliable operation; the unsatisfactory *ad hoc* manner of combining ID values; and the lack of robustness to illumination variations across the scene. The system had in addition proved extraordinarily complex to debug. To work out why processing had failed in a false positive or false negative scenario, entailed looking inside the code to see what was going wrong. Each incremental improvement in the system, was necessarily preceded by such a laborious debugging exercise.

The circular targets used have $64 = 2^6$ distinct forms with IDs ranging from 0 to 63. Pasting one fiducial target per ceiling tile (of dimensions 60 cm $\times$ 60 cm) in our laboratory gives the ability to uniquely determine location within an area of $0.6 \times 0.6 \times 64 = 23.04$ square metres. However, the laboratory is much bigger than this. It would be possible to move from the original system using six rings to perhaps eight or ten rings *i.e.* 256 or 1024 targets. Unfortunately, the more rings the narrower they get and the more difficult they are to find without error.

5.6 The Hough Transform

As we are now looking for circles within a scene, the Hough Transform [59] suggested itself, as this is a technique to find geometric primitives within noisy (real world) data. The basic idea is to transform the image to the n-dimensional parameter space for the requisite primitive. Locations of local maxima in the parameter space, then give likely parameter values for such primitives as exist in the original image. Hough transforms are traditionally performed on edge data, as these are regarded as holding the shape information. Let us first consider a Hough Transform to find straight line segments within an image. Edge pixel (x, y) may be part of a straight line $y = mx + c$. While an infinite number of straight lines may pass through this point, the lines are constrained to a set where c and m are related

$$c = y - mx$$

Now a plot of possible (m, c) values in (m, c) space forms a straight line. If we accumulate these lines in a 2D accumulator array, and work through all (x, y) positions in feature space (*i.e.* typically all edge pixels), then we can look for local maxima in (m, c) space . A point in (m, c) space corresponds analogously to a line in (x, y) space, and *vice versa*. This is the idea behind the Hough Transform, though the accumulator space may in general have a higher number of dimensions. Naturally a 3D parameter space at any significant resolution presents an immediate storage problem! The Hough Transform algorithm may be more generally described:

- Find all feature points in an image
- For each feature point
  - Find the possibility that the primitive with a particular set of parameters passes through the feature point.
  - Increment each element of the accumulator by that possibility
- Find the local maxima in the accumulator

The Hough Transform for a circle has a particularly intuitive geometric interpretation. For each feature point in our application we know both the position and the gradient *i.e.* we have additional knowledge to restrict the possible set of parameters. At this stage, let us try just to find the location of the circle rather that its radius, so we are only working with 2D parameter space (x_0, y_0) instead of (x_0, y_0, r),

where the equation of the circle is given by:

$$(x - x_0)^2 + (y - y_0)^2 = r^2$$

The gradient and position of a feature point defines a line in parameter space on which the centre of the circle must lie. This is because:

1. the edge of a circle is at right angles to its diameter, and
2. the gradient of an intensity field is at right angles to the detected edge.

It is noteworthy that parameter space and image space in this technique are one and the same! Figure 5.22 shows lines of gradient building up in the accumulator buffer. As the perpendiculars to a circle intersect at the same point (namely its centre), the darkest pixels indicate the highest probability of location of the target's centre. Clearly, a more elaborate technique can be used if it were required to find the circles' radii, or indeed if ellipses (with one more parameter) were to be found. However, in the spirit of using the simplest technique first, the circular Hough Transform was used to determine the location of a set of targets that were square-on and hence circular to the camera, The outcome from the Hough method shows that local maxima at the centre of the circles emerge clearly (Figure 5.24). However, the problem of locating a target has been transformed into locating high intensity clusters in a noisy background field. One could use thresholding to filter the intensity clusters or to limit the number of matches in the scene. However, this is messy and one location problem has been turned into another harder location problem! It was realised that as we have total control over the target design and the image quality is itself good from the camera used, that there is no need to move to the Hough Transform, which will come into its own in picking out naturally occurring objects with details that correspond to geometric primitives. The Hough Transform was thus abandoned, with an enhanced belief in really choosing something special for a target to make detection as easy as possible.

Out of interest, instead of just using the edge pixels, every pixel in the gradient image was used to contribute to the accumulator buffer, weighted naturally by the intensity of the gradient at that pixel. The outcome Figure 5.25 was little different, still showing the same intensity peaks, but with slightly greater contrast against the background. The latter approach is therefore to be preferred, despite the fact it goes against the "textbook". There are technical reasons to also favour this approach: it avoids the use of thresholding before the Hough transform, and also draws from a larger set of source data *i.e.* the entire image. Ironically, the location techniques for circular targets, also works perfectly for

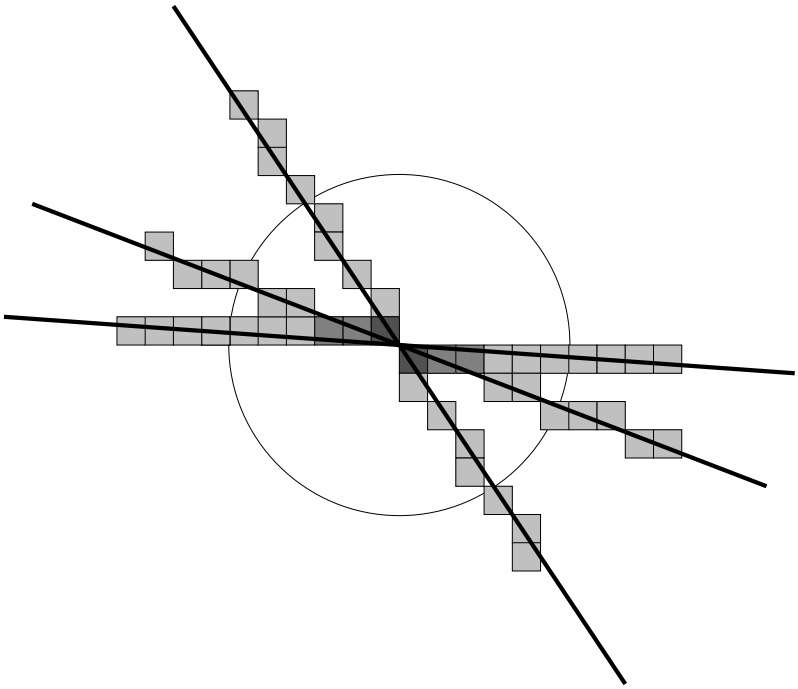

Figure 5.22: Accumulator buffer

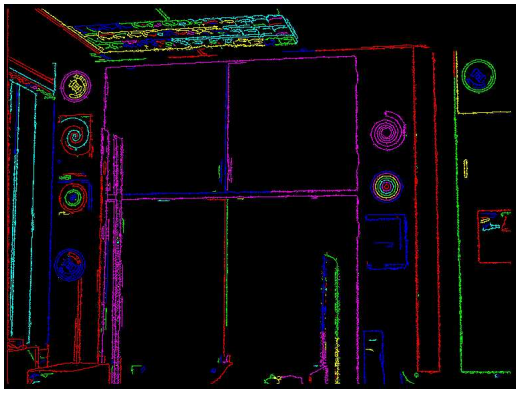

Figure 5.23: Scene edges

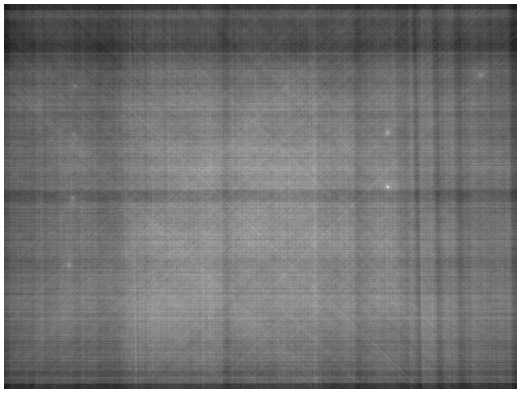

Figure 5.24: Hough processed using edge data

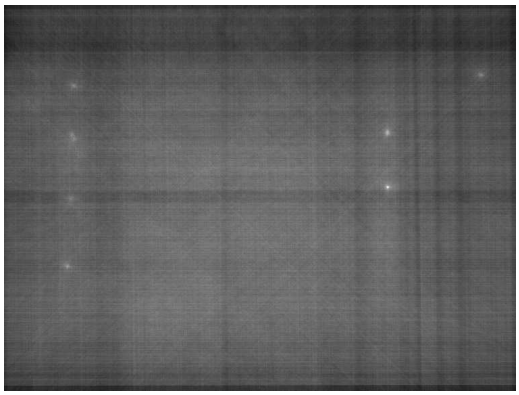

Figure 5.25: Hough processed using full scene data

spiral targets, as the perpendiculars also intersect at a point. However, circular techniques involving a radius parameter would fail, as a spiral in this context may be thought of as a circle with an ever varying radius.

5.7 RAG Target Generation

The idea suggested itself of not just using the RAG of a target as verification but as a means of detection and encoding an identifier. The design considerations are:

detectability The RAG of a target should have a property which distinguishes it from the background. This is designated a key RAG, and the decision was made to give each target the same key RAG. After turning the scene into a RAG, detecting targets is equivalent to a graph search for the key

RAG. Many instances of this key may be found, each corresponding to a visible target. In particular, the key RAG consists of an anchoring black region at the outside of each target. Within this black region are four white regions at the corners of the target. Within each of these white regions in turn are zero, one, two and three small black blobs. It is unlikely that this tree structure, admittedly totally naïve in conception, will naturally occur in any scene. Note that the key RAG is actually a key tree (i.e there are no loops in the graph). There are two individually sufficient reasons for this:

(i) a region graph with only two colours can only be a tree

(ii) recursive subdivision of regions is a simple way to produce RAG graphs. The technique, however, only generates trees.

Initial RAG target designs took the anchoring black region as close to the edges of the (A4) sheet of paper as possible to maximise the size of the target design. After problems involving the merging of this black region and other darker regions outside the target area, it was realised that a generous white margin was required.

distinguishability Each target must have a different RAG, which allows it to be distinguished. This is called an identifier or ID RAG. Note that visually distinguishable patterns may have the same RAG. Now one could conceive of a RAG grammar, where all targets conform to the RAG grammar. Following the RAG grammar is what distinguishes the target from the background, and it is the nature of the RAG sentence expressed therein that distinguishes one target from another. It was felt that this approach would be overly complex, and that simple graph searching for a particular RAG would be a taxing enough task!

In particular, the central white square region in every target is the top node of the ID RAG tree. An ID RAG has to be a tree for the reasons given above. The algorithm used to generate different ID RAGs, is simply to produce all RAG trees starting with a single white node, by cumulatively adding single nodes in all possible ways, up to a certain node count. Clearly, there are physical representation constraints on this process. For example, a white region with a million enclosed black blobs would be a valid theoretical ID RAG, but the black blobs would have to be below a practically detectable size if the target were ever to be physically realised.

diversity The number of different targets that can be produced with different RAGs is a complex

issue and is dealt with later.

locatability The circular targets only provide one locatable point i.e the centre. The RAG approach frees us from this design constraint, to provide more locatable points per target. The more locatable points, the more reliable will be the positioning information derived by the system. One should take maximum advantage of the size extent of any target for greatest positional accuracy and this suggests putting locatable points near the edge. The locatability information should itself have a meaningful form in a RAG representation for a unified approach. There could be some form of doubling up of functionality, and the locatability information could be part of the key RAG or ID RAG. As the key RAG is fixed, it was simpler to include the locatability information within this and for best locatability to then place the key RAG on the outside of the target. The key pattern on the target has no rotational or reflective symmetry allowing the orientation of the target to be determined without ambiguity. The centres of the white regions within the key RAG, are the four location points. These are near to the corners of the target, and each can be distinguished due to its unique RAG characteristics.

5.7.1 Production Algorithms

Consider the set of distinct RAGs that can be drawn by colouring the squares in an 11×11 grid, say, either black or white. This is a crude but effective way of ensuring a minimum target feature size. Grid colourings can be generated:

1. bottom-up in grid space

   The first stage is to generate all colourings of an 11×11 grid. The RAGs for these $2^{11^2} \approx 2.7 \times 10^{36}$ colourings are derived, and targets with duplicate RAGs are discarded.

2. bottom-up in RAG tree space recursively

   Sub-trees are combined from the bottom-up. Those where the corresponding target would be too large to fit an 11×11 grid are removed.

3. top-down in RAG tree space recursively

   To produce n subtrees a given area is evenly subdivided into n parts.

The combinatorial explosion of 1 renders it impractical for generating targets. Both 2 and 3 will generate targets, but 3 does not intelligently use area. For a deep tree, the area will too readily be

subdivided repeatedly to a size below that considered resolvable. 2 more intelligently tries to pack sub-targets together (naturally into as small an area as possible) and so was chosen. The success of this approach, which directly equates to the number of targets that can be drawn, is dependent upon the cleverness with which sub-targets are combined.

Simple target joining and target drawing algorithms were used to save time. By doing this, a number of possible targets that could have otherwise been drawn, were missed out. However, the question is not how many targets can be produced, but whether a particular method can produce enough. Practical findings have demonstrated the "enough". Some additional work was undertaken to explore the "how many", and in addition to determine the scalability of the method. The simplifications made for method 2 were:

(a) all targets are drawn as rectangular

(b) pack n targets by packing two (namely the two smallest) at a time

(c) pack 2 targets by aligning their longest sides

5.7.2 Generating the Graph

As the target generation code is stand-alone, there is no necessity for it to be integrated with the 'C' code being used for the rest of the performance critical system, and hence it can be written in a more suitable language. The functional language ML was chosen. A tree is defined recursively in the ML code below as a root node associated with a colour and a list of sub-trees. The colour is represented as an integer.

```
type Colour   = int; (* 1 represents black, 0 white *)
datatype Tree = Node of Colour * Tree list;
```

Note that this simplistic representation is quite powerful enough for our purposes. We define an equality operator on trees. Two trees are equal if their root nodes are equal in colour, and their subtree lists contain the same elements (though not necessarily in the same order). A quick way of determining that two trees are not equal is to see if the lengths of their subtree lists are different, and this is used in the code for potential efficiency savings.

```
fun tree_equal (Node(a,al)) (Node(b,bl))
=
if ( ( a = b ) andalso ( length al = length bl) )
then lists_equal tree_equal al bl
else false;
```

This code is based on a list equality operator `lists_equal` which takes an equality operator and the two lists as arguments. A `Rule` operator is defined which adds a node to a tree in all possible positions. The output is a list of all possible resultant trees. When we add a node to a tree, we can add it at the top level *i.e.* add a new daughter or we can add the node to one of the existing daughters. The function to add the node to one of the daughters is called `Rule_list` , which returns a list of daughter lists, each of which reflects a node added to each daughter in all possible ways. Note that `Rule` and `Rule_list` must be defined simultaneously using the `and` construct as they are mutually recursive.

```
(* Rule: Tree -> Tree list
   extends tree by one Node in all possible ways *)


fun Rule (Node(c, tl)) =
let

        val l1 =  Node(c, Node( 1 - c, [] )::tl );
                  (* add a new daughter - note colour change *)

        val l2 = map ( fn e => Node(c,e) ) (Rule_list tl);
                  (* re-attach top node to each possible way of extending
                     existing daughters

                     map applies a function to each element of a list

                     fn e => Node(c,e) is an inline function definition
                  *)

in
        l1::l2  (* add l1 to head of list l2 *)
end

(* Rule_list : Tree list -> Tree list list
                extends each list in all ways by adding a single node *)

and Rule_list []       = []
 |  Rule_list (h::t)   = ( map ( fn a => (a::t) ) (Rule h)  )
                         (* apply rule to 1st daughter and
                            re-attach all results to existing tail *)

                            @ (* list catenation operator *)

                         ( map ( fn a => h::a) (Rule_list t) ) ;
                         (* leave 1st daughter alone and
                            re-attach to all results of recursive call *)
```

With the ability to grow a tree by a single node, the next stage is to apply this operation repeatedly to a seed tree to produce a set of distinguishable trees to work as targets. The idea is to apply `Rule` up to a maximum of 8 times (say): the function which does this is called `Rules` (note the plural) which takes a single maximum depth argument. Figure 5.26 shows `Rules` is implemented in terms of a recursive function `Rules0`, which is initially also given a starting depth of 0, and the seed tree `Node(1,[])`. Note that `uniq_tree` is called twice to remove duplicate trees instead of once: why? Well, performance was experimentally determined to increase by removing duplicates as soon as possible, to stop unnecessary combinatorial explosions. It was realised that even a simple call of `Rule` will produce duplicates. In Figure 5.27 two different construction operations lead to indistinguishable trees.

5.7.3 Rendering the Graph

The tree information must be turned into a form which is suitable to render. Each tree node must be drawn as a region, and the most straightforward way of achieving this is to associate each node with a rectangular area with x and y offsets, width and height. Each tree is rendered within a rectangular region, bounded by the box of its root node. To hold this representational information, we define a new type of tree, a DrawTree:

```
type Box          = real * real * real * real; (* x, y, width and height *)
datatype DrawTree = DrawnNode of Colour * DrawTree list * Box;
```

To render a tree (in effect to turn a `Tree` into a `DrawTree`) a recursive algorithm is used. We render a leaf node as a unit box, and render a parent node, by combining the renderings of its daughter nodes, using the combine function.

```
(* draw_tree : Tree -> DrawTree *)
fun draw_tree (Node( c, [])) = DrawnNode(c,[], (0.0,0.0,1.0,1.0) )
 |  draw_tree (Node( c, t )) = combine c (map draw_tree t);
```

The `combine` function joins together a set of rectangles, and creates a bounding rectangle round these components suitably arranged. Clearly, to obtain as great a variety of trees within a given area as possible, these component rectangles should be packed as space-efficiently as possible. This is the classic box packing problem in 2D. A brief investigation was made into optimal packing strategies, and it was decided for simplicity to pursue a naïve algorithm that was expected to achieve a reasonable

```
(*       Rules0:      Tree list -> int -> int -> Tree list

         Arguments:   tree_list  - input list of distinct trees
                                   with depth "nodes" added

                      depth      - number of nodes added on so far

                      max_depth  - maximum number of nodes to be added

         Returns:     outputs list of distinct trees
                      with depth to max_depth "nodes" added
 *)

fun Rules0 tree_list depth max_depth =
tree_list @     (* pre-pend argument trees to result list *)
(
if ( depth >= max_depth )
then
        []      (* reached maximum number of nodes to be added *)
else
        let
                val l = map ( uniq_tree o Rule) tree_list;
                (*
                   turn each tree into a list of trees one size bigger

                   remove duplicates in each of these lists using
                   uniq_tree: Tree list -> Tree list function which
                   is written in terms of the tree equality operator

                   o is the function catenation operator
                 *)

                val tree_list' =  uniq_tree (flatten l);
                (*
                   turn list of lists into a single flat list using
                   flatten: 'a list list -> 'a list function

                   remove any duplicates in this new generation of trees
                 *)
        in
                Rules0  tree_list' (depth+1) max_depth
                (* recursive call *)
        end
);

(*       Rules : int -> Tree list

                user interface (and entry point to)
                recursive function Rules0

         Argument:    max_depth  - maximum number of nodes to be added

         Returns:     outputs list of distinct trees
                      with 0 to max_depth "nodes" added
 *)

fun Rules max_depth = Rules0 [ (Node(1,[])) ]  0 max_depth;
```

Figure 5.26: Graph generating code

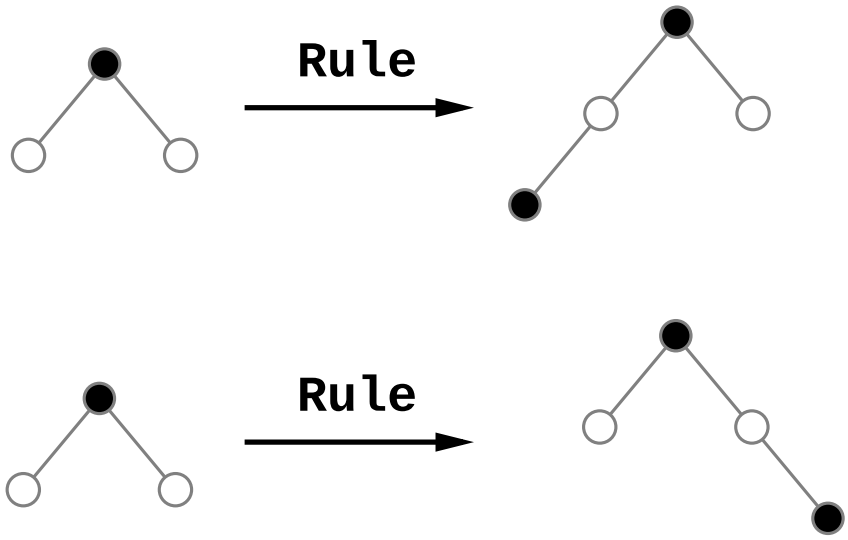

Figure 5.27: Generative step in producing graphs: function Rule is shown adding a node in two ways

packing density. At the end of the day, a sufficient number of different targets were produced, so it did not prove necessary to go back and revise the simplistic approach. The basic approach is to sort the component rectangles in order of size, using an ordering relationship.

```
(* get_area : DrawTree -> real *)
fun get_area (DrawnNode(c, tl, (x,y,w,h))) = w * h;(* area of node *)

(* orderop : DrawTree -> DrawTree -> bool *)
fun orderop a b = (get_area a) < (get_area b);     (* area ordering relationship *)
```

The two smallest rectangles will be combined together, to reduce the size of the problem by one. Combining smaller rectangles together first is expected to achieve the densest packing. Basically the problem of combining N rectangles (using `combine`) is reduced to that of combining 2 rectangles (using `combine2`). If there is only one rendered daughter sub-tree (i.e there is either a single daughter node or the rendered daughter nodes have been combined into one `DrawTree`), then a rectangle is drawn round the daughter's rectangle including a frame one unit wide. From the way the (x,y) coordinates are set up it is apparent that these are relative to the parent node, at this stage, rather than absolute.

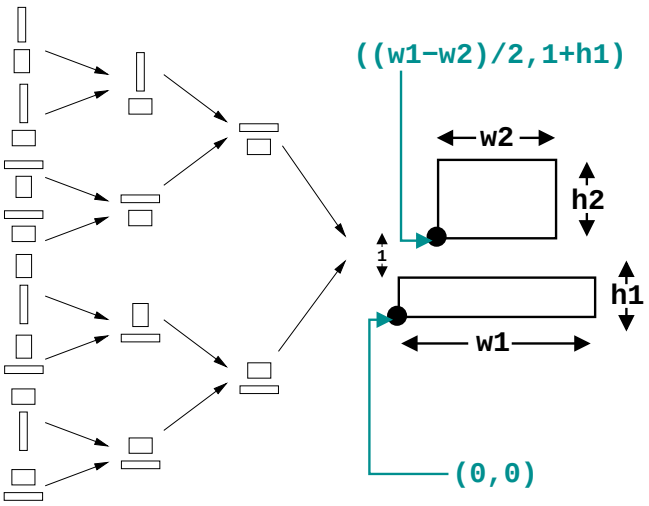

Figure 5.28: Space efficiently combining rectangles

```
(* combine: Colour -> DrawTree list -> DrawTree
                    takes a list of rendered trees and combines them into a single
                    rendered tree with top node of colour c *)
fun combine c l =
let
        val l' = sort orderop l;  (* sort rectangle list in terms
                                     of size with smallest first *)
in
        case l' of [] => raise InvalidOp (* error condition *)

                  (* draw surrounding box using a frame of one unit *)
                  | (DrawnNode(c1,l1, (x1,y1,w1,h1) ))::[]  =>

                  DrawnNode(
                                c,
                                [ DrawnNode(c1,l1,(1.0,1.0,w1,h1) ) ],
                                (0.0,0.0,w1+2.0,h1+2.0)
                           )
                (* combine two at a time *)
                |    a::b::t => combine c ( (combine2 c a b):: t )
end;
```

The problem is solved apart from combining two rectangles. For densest packing, the strategy used is to place the longer sides of the two rectangles against each other and to centre them. In this way compound shapes are regularised into being as square as possible, rather that being awkward long thin shapes. To simplify the coding the rectangles are flipped if necessary to become horizontally oriented and presented to the underlying combining routine `Combine2`, so that the wider rectangle is presented first. Figure 5.28 shows graphically how two rectangles are combined.

```
(*      combine2 : Colour -> DrawTree -> DrawTree -> DrawTree

        combines two rendered targets to
        form a composite rendered target
 *)

fun combine2 (c:Colour) (a:DrawTree) (b:DrawTree) =
let
        val DrawnNode(c1,l1, (x1,y1,w1,h1) ) = a;
        val DrawnNode(c2,l2, (x2,y2,w2,h2) ) = b;
in
        if (w1 < h1 )
        then
                (* reflect flips a shape around the y = x axis *)
                combine2 c (reflect a) b
                (* reflect so 1st rectangle wider than high *)
        else
                if ( w2 < h2 )
                then
                        combine2 c a (reflect b)
                        (* reflect so 2nd rectangle wider than high *)
                else
                        if ( w1 < w2 )
                        then    combine2 c b a   (* put wider one first *)
                        else    Combine2 c a b
end;

(*      Combine2 : Colour -> DrawTree -> DrawTree -> DrawTree

        combines two rendered targets to form a composite
        rendered target only horizontally oriented targets
        are allowed: the wider must be first
 *)

fun Combine2 (c:Colour) (a:DrawTree) (b:DrawTree) =
let
        val DrawnNode(c1,l1, (x1,y1,w1,h1) ) = a;
        val DrawnNode(c2,l2, (x2,y2,w2,h2) ) = b;
        val a' = DrawnNode(c1,l1, (0.0,0.0,w1,h1) );
        val b' = DrawnNode(c2,l2, ((w1-w2)/2.0,h1+1.0,w2,h2) );
in
        DrawnNode(c,[a',b'], (0.0,0.0,w1,h1+h2+1.0) )
end;
```

A final pass converts the relative coordinates to absolute ones: dx and dy represent the cumulative x and y offsets respectively which are initialised to 0.0.

```
fun relative_to_absolute0 dx dy (DrawnNode( c, [], (x,y,w,h) ))
        = DrawnNode( c, [], (x+dx,y+dy,w,h) )
 |  relative_to_absolute0 dx dy (DrawnNode( c,  l, (x,y,w,h) ))
        = DrawnNode
          (
                  c,
                  map (relative_to_absolute0 (dx+x) (y+dy) ) l,
                  (x+dx,y+dy,w,h)
          );


val relative_to_absolute = relative_to_absolute0 0.0 0.0;
```

The overall function to convert a `Tree` to a `DrawTree`, can then be expressed as `Draw_Tree`:

```
val Draw_Tree = ( relative_to_absolute o draw_tree);
```

5.7.4 Conversion to PostScript

The set of `DrawTree`s created from the set of `Tree`s contains all necessary rendering information, however, it is not yet in a recognised computer graphics format. The format chosen was PostScript as this allows direct rendering by the rasterising engine of a printer at high resolution. The syntactic sugar of conversion to PostScript is outside the main thrust of discussion. However, a final filter of the `DrawTree`s is necessary to ensure that the minimum feature size does not drop below a certain level. The leaf nodes are squares of unit side, and the notional maximum grid on which regions can be placed is 11×11. The appropriate filter function is:

```
fun small_enough (DrawnNode(c,dl,(x,y,w,h))) = (w <= 11.0) andalso  h <= 11.0 );
```

The ID trees that pass the filter are scaled to the region available for them on the target, before the final conversion to PostScript. The expression of the PostScript document, with a different target per page, is therefore:

```
map
(
        draw_tree_to_ps
        (* map DrawTree to PostScript *)
        o
        ( scale_draw_tree 5.5 5.5 1.25 3.0 )
        (* scale size of DrawTree to window available on page *)
)
( filter small_enough (map Draw_Tree (Rules n)) );
(* generate DrawTrees removing those with too fine details *)
```

5.7.5 Generating ‘C’ Static Data

A similar post-processing phase turns the selected ID trees into a unique string representation that is suitable for handling within a ‘C’ program. See Figure 5.55. Tree equality is equivalent to string equality of these strings. To load ‘C’ tree structures directly from file is awkward: both in terms of coding and defining the relevant file formats. This is why a simple string representation was chosen.

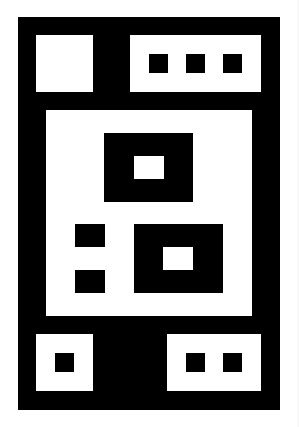

Figure 5.29: Target 22

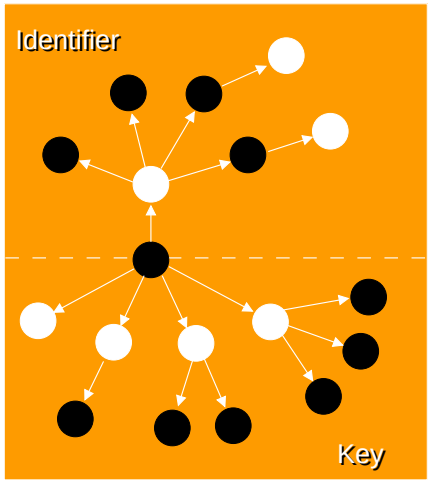

Figure 5.30: RAG of target 22

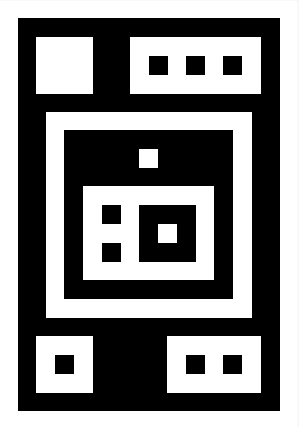

Figure 5.31: Target 294

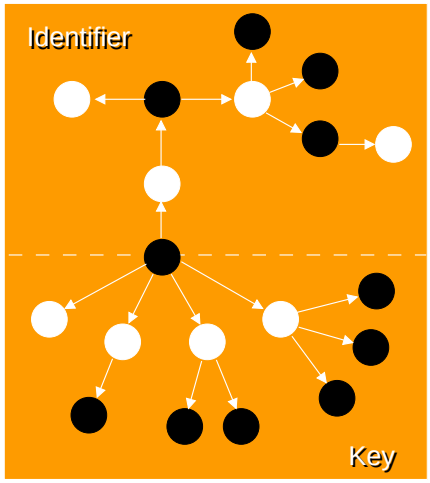

Figure 5.32: RAG of target 294

No of Graphs			
Level	Nature of Trees Removed		
	None	Duplicates	Duplicates then Outsized
0	1	1	1
1	2	2	2
2	4	4	4
3	10	8	8
4	34	17	17
5	154	37	37
6	874	85	75
7	5914	200	130
8	46234	486	191
9	409114	1205	252
10	ℵ	3047	306
11	ℵ	7813	350
12	ℵ	20299	386
13	ℵ	ℵ	ℵ

Table 5.3: Tree diversity with level

5.7.6 Computational Tractability

Early versions of the target generation software took 3 hours to compute a set of 306 targets. Through judicious application of uniqueness filters, execution time was reduced by a factor of 60. Though runs of the order of minutes are acceptable to produce a one-off set of targets, they limit the extent to which the scalability of target generation can be investigated. In Table 5.3 an ℵ represents the stage at which there is a combinatorial explosion *i.e.* the result does not arrive within a tractable time. Actually though the results are experimental, it apparent that the formula for the n^{th} term in column 2 is $\sum_{i=0}^{n} i!$ — a result which can be derived theoretically. Knuth [60] provides an evaluation strategy for a_n, the number of topologically distinct trees with n nodes, which corresponds to the difference between the rows of column 3, see Section 5.7.7.

With duplicates removed the computation can proceed a couple of levels further without exploding. Removing outsized graphs (which render with insufficiently large detail on paper as they have to be shrunk to fit the area available within the target) does not change the location of the combinatorial explosion, because it occurs before the size filter is applied. Unfortunately, using the method described above it is impossible to reach the limit on the number of targets drawable on an A4 sheet.

The upper limit on the value in column 4 in Table 5.3 is not clear. A graph shows the curve is starting to level-off at around level 12, but it is not yet "round the bend". One insight for computability is to take the size filter at the end of the pipeline, and to use it in the generative process itself, with the aim of cutting down any combinatorial expansion in the early stages. Though the semantic clarity of the pipeline is compromised, the modified software allows the scalability characteristic of target generation to be nailed down.

5.7.7 Number of Topologically Distinguishable Trees

The number of topologically distinct trees with n nodes, a_n is given by the recursive formula:

$$a_n = \sum_{j_1+2j_2+\ldots=n-1} \binom{a_1+j_1-1}{j_1} \cdots \binom{a_{n-1}+j_{n-1}-1}{j_{n-1}} \text{ where } \binom{n}{r} \equiv {}^nC_r \qquad (5.3)$$

Assuming there are j_k subtrees with k vertices then we may choose j_k of a_k possible k vertex trees. Note that a particular node has been designated as the root node. This is the case with our RAG targets as the parent relationship between regions is well defined: a parent node surrounds a daughter node. Clearly, if any node in the graph could act as the root, then there would be even fewer distinct graphs. The ML implementation of formula 5.3 is shown in Figures 5.33 and 5.34. With the standard numeric implementation of the binomial coefficient function, overflow occurs at the exceptionally low order term of a_7. Moreover, the calculation is atrociously inefficient as the last term that can be computed in under a minute is a_8. Excessive combinatorial explosion occurs after this stage. Using the generating function:

$$A(z) = \sum_n a_n z^n \quad \text{with} \quad a_o = 0$$

and the mathematical identity:

$$\frac{1}{(1-z^r)^a} = \sum_j \binom{a+j-1}{j} z^{rj}$$

it is possible to show that:

$$A(z) = \frac{z}{(1-z)^{a_1}(1-z^2)^{a_3}(1-z^3)^{a_3}\ldots}$$

```
(*
 * J0 : int -> int -> int list list
 *
 *         left:        remainder of sum to be made up
 *
 *         k:           highest value of k allowed to contribute
 *                      terms of form k * j[k] to sum
 *
 *
 *  returns list of all lists of form  [ j[1], j[2], ... , j[k] ]
 *  such that j[1] + 2 * j[2] + ... + k * j[k] = left
 *
 *  Utility functions used:-
 *
 *        flatten       - turns a lists of lists into the obvious list
 *        seq 1 n       - makes list [ 1, 2, 3, ..., n ]
 *)

fun J0 left k =

if ( k = 1 )
then
      [[left]]
      (* last k value - no choice but to make j[1] complete the sum *)
else
      flatten
      (
            map
            ( fn j => map (fn l => j::l) ( J0 (left - j * k) (k-1) ) )
            ( seq 0 (left div k) )
      );

      (*
       * j[k] has possible values 0 to left/k
       * add j[k] to beginning of lists produced by recursion
       * where:
       *      remaining sum is reduced by j[k] * k
       *      looking for terms j[k-1]
       *)
(*
 * J :  int -> int list list
 *
 *              n : single integer argument
 *
 *  returns a list of all lists of form [ j[1], j[2], ..., j[n-1] ]
 *  such that j[1] + 2 * j[2] + ... + (n-1) * j[n-1] = (n-1)
 *
 *  calls recursive function J0
 *  lists are reversed as they are generated backwards for efficiency
 *  (both computational and reducing number of arguments)
 *  j[n-1] is determined first and so on, with j[1] determined last
 *)

fun J n = map rev ( J0 (n-1) (n-1) );
```

Figure 5.33: Implementation of j_n in formula 5.3

```
(*  choose : int -> int -> int
 *
 *  Calculate binomial coefficients. The obvious implementation
 *  fun choose m n = factorial(m) div ( factorial(n) * factorial(m-n) );
 *  could not be used as integer overflow occurred early i.e. in computing a[7]
 *  Utility functions used:-
 *           list_intersection  - operation clear from name
 *           list_minus         - operation clear from name
 *           multiply_list      - operation clear from name
 *)

fun choose m n =
if ( n = 0 ) orelse (m = n) then 1
else let
         val numer =   seq 1 m;
         val denom = ( seq 1 n ) @ seq 1 (m-n);
         val intersection = list_intersection numer denom;
         val numer_val =  multiply_list (list_minus numer intersection);
         val denom_val =  multiply_list (list_minus denom intersection);
     in
         numer_val div denom_val
     end;

(*
 * term: int -> int -> int
 *
 * given j[k] calculate kth multiplicative term in expression for a[n]
 *)

fun term k jk = choose (  (a k) + jk - 1 )  jk
and
(* a: int -> int
 *
 * calculate a[n] less efficiently, n is single integer argument
 * Utility functions used:-
 *      add_list      - operation clear from name
 *      multiply_list - operation clear from name
 *      zippair       - "zips-up" two lists using a binary operation
 *      pair          - combines its two arguments into a single 2-tuple
 *)

a n =
if ( n <= 1 ) then n
else let
         val NS = seq 1 (n-1); (* list with value    [ 1, ..., n-1 ] *)
         val JS = J n; (* list holding lists of form [ j[1], ..., j[n-1] ] *)
     in
         add_list
         (
                map
                ( multiply_list o (zippair term) )
                ( map (pair NS) JS )
         )
         (* a copy of NS is paired up with every member of JS; corresponding
          * elements within each pair are used to compute multiplicative
          * terms which are multiplied together; and then finally added
          *)
     end;
```

Figure 5.34: Implementation of a_n in formula 5.3

However, this is hardly a convenient computational form for the generating function (given the denominator is an infinite series). The generating function can be shown as:

$$A(z) = z + z^2 + 2z^3 + 4z^4 + 9z^5 + 10z^6 + 48z^7 + 115z^8 + 286z^9 + 719z^{10} + 1842z^{11} + \ldots$$

More useful relationships to compute the values of a_n are:

$$na_{n+1} = a_1 s_{n1} + 2a_2 s_{n2} + \ldots + na_n s_{nn}$$

Define:

$$S_{nk} = \sum_{1 \leq j \leq \frac{n}{k}} a_{n+1-jk}$$

This is useful to calculate a_n since

$$S_{nk} = S_{(n-k)k} + a_{n+1-k} \tag{5.4}$$

The corresponding ML code follows:

```
(*  A: int -> int
 *  Computes a[n] more efficiently, n is the single integer argument
 *)

fun A 0 = 0
 |  A 1 = 1
 |  A n = (
                    add_list
                    (
                             map  (fn j => j * (A j) * (S (n-1) j)) (seq 1 (n-1))
                    )
          )
          div
          ( n - 1 )

and

(* S: int -> int -> int
 * Computes S[n][k],  n, k are integer arguments
 *)

S n k = if    ( ( n - k ) > k )
        then
                ( S (n-k) k ) + ( A (n+1-k) )
        else
                add_list
                (
                        map (fn j =>  A (n + 1 - j * k)) (seq 1 (n div k))
                );
```

n	a_n	$\sum_{i=0}^{n} a_i$
0	0	0
1	1	1
2	1	2
3	2	4
4	4	9
5	9	17
6	20	37
7	48	85
8	115	200
9	286	486
10	719	1205
11	1842	3047
12	4766	7813
13	12486	20299
14	32973	53272
15	87811	141083
16	235381	376464

Table 5.4: Calculated values of a_n (and $\sum_{i=0}^{n} a_i$) using Knuth formula

This calculation method for a_n is much more efficient (the function is called A to distinguish it from the less efficient a). The last term that can be computed in under a minute is a_{14}. Notably when the efficiency hack in equation 5.4 suggested by Knuth is removed, there is no difference in performance with a_{14} taking 30 seconds to compute before and after this expression's removal. Note also, that this expression cannot be used in all circumstances because $(n - k) > k$ must be true. The values of $\sum_{j=0}^{n} a_j$ computed in Table 5.4, agree with those experimentally derived in the third column of Table 5.3.

5.7.8 Software Modification for Computational Tractability

Due to the daunting nature of placing the later stage of the pipeline within an earlier stage, there was considerable surprise at the ease with which ML permitted this change. It was literally a 1.5 line change: adding one new line and changing a variable in another.

```
fun Rules0 tree_list depth max_depth =
tree_list @
(
if ( depth >= max_depth )
then
      []
else
      let
            val l           = map (uniq_tree o Rule) tree_list;

            val tree_list'  = uniq_tree (flatten l);
                              (* initial uniqueness filter *)

            val tree_list'' = filter (small_enough o Draw_Tree) tree_list' ;
                              (* additional size filter - new line *)
      in
            Rules0  tree_list'' (depth+1)  max_depth
            (* change is using tree_list'' here instead of tree_list' *)
      end
);

fun Rules max_depth = Rules0 [ (Node(1,[])) ]  0 max_depth;
```

With this increased pruning, the 4^{th} column of Table 5.3 can be taken further as shown in Table 5.5. The resulting S-shaped curve is shown in Figure 5.35. Note that the maximum number of targets was now reached with no combinatorial explosion, at level 18, within a second; in fact processing was too quick to measure. Note that previously the three hour processing time occurred as early as level 10. In fact, there will no form of combinatorial explosion lying in wait after level 18, as all further trees have been pruned!

The tractable processing can be justified in an interesting *a posteriori* argument. As the aim was a set of targets which could be stuck to the ceiling (in finite time) the processing, when limited by some such form of physical constraint, should itself occur in finite time. Computer output that is to be useful sets limits on its own tractability of computation. In short, one should have been suspicious of the combinatorial explosion of the process (sets of up to 20299 targets being involved) when it was known that only around 500 targets would ultimately be selected. For once, it has been possible to take the target generation code to its natural conclusion which is often difficult with software in practice. The final version produces all 449 targets used in practice in a fraction of a second and the elegance of the code has been optimised.

Of course, the packing algorithm is far from optimal, and though it would be desirable to enhance this (the number of possible targets would then probably exceed 500 for a grid size of 11×11) it is not something that was ever genuinely planned. It is possible to observe the relative success of the naïve packing algorithm in Figure 5.36 which shows the target appearing on the last page of the PostScript

Level	Total Number of Graphs
0	1
1	2
2	4
3	8
4	17
5	37
6	75
7	130
8	191
9	252
10	306
11	350
12	386
13	411
14	427
15	438
16	444
17	448
18	449
19	449
20	449

Table 5.5: Number of graphs with node count for an 11×11 grid

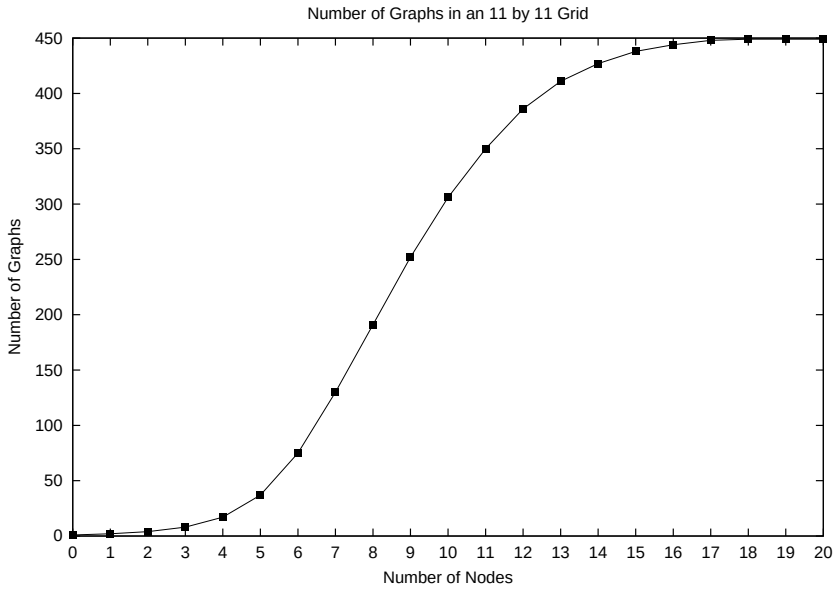

Figure 5.35: Number of graphs against number of nodes for an 11×11 grid

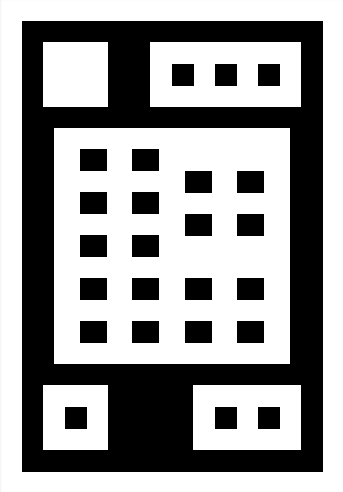

Figure 5.36: Last page in target document

document. The algorithm has not quite managed to cram in a 5×5 array of black dots, which would be the maximum number possible, but it almost got there.

5.7.9 Scalability with Target Size

The previous scalability investigated was with the number of nodes in the RAG tree for a fixed grid size of 11×11. For a 9×9 grid the number of possible targets is low enough to collate them in a single diagram (Figure 5.37): the visible progression in ID graph complexity is clear. For other grid sizes, Tables 5.6 and 5.7 show the resulting number of targets. Table 5.7 factors out the more significant scalability of the method and is unconcerned with the steps in target production *i.e.* the dependency on the number of nodes added. The logarithmic and non-logarithmic graphs corresponding to the data in Table 5.6 are Figures 5.39 and 5.38 respectively.

5.7.10 Conclusions

Some artifact of the method does not increase the number of targets when the grid size increases from $n \times n$ to $(n+1) \times (n+1)$, where n is odd. As a larger area becomes available to draw the target, one would expect a great increase in the number of possible targets. The figures show this — the explosion in numbers of targets with size is why a logarithmic plot is used to show trends. The

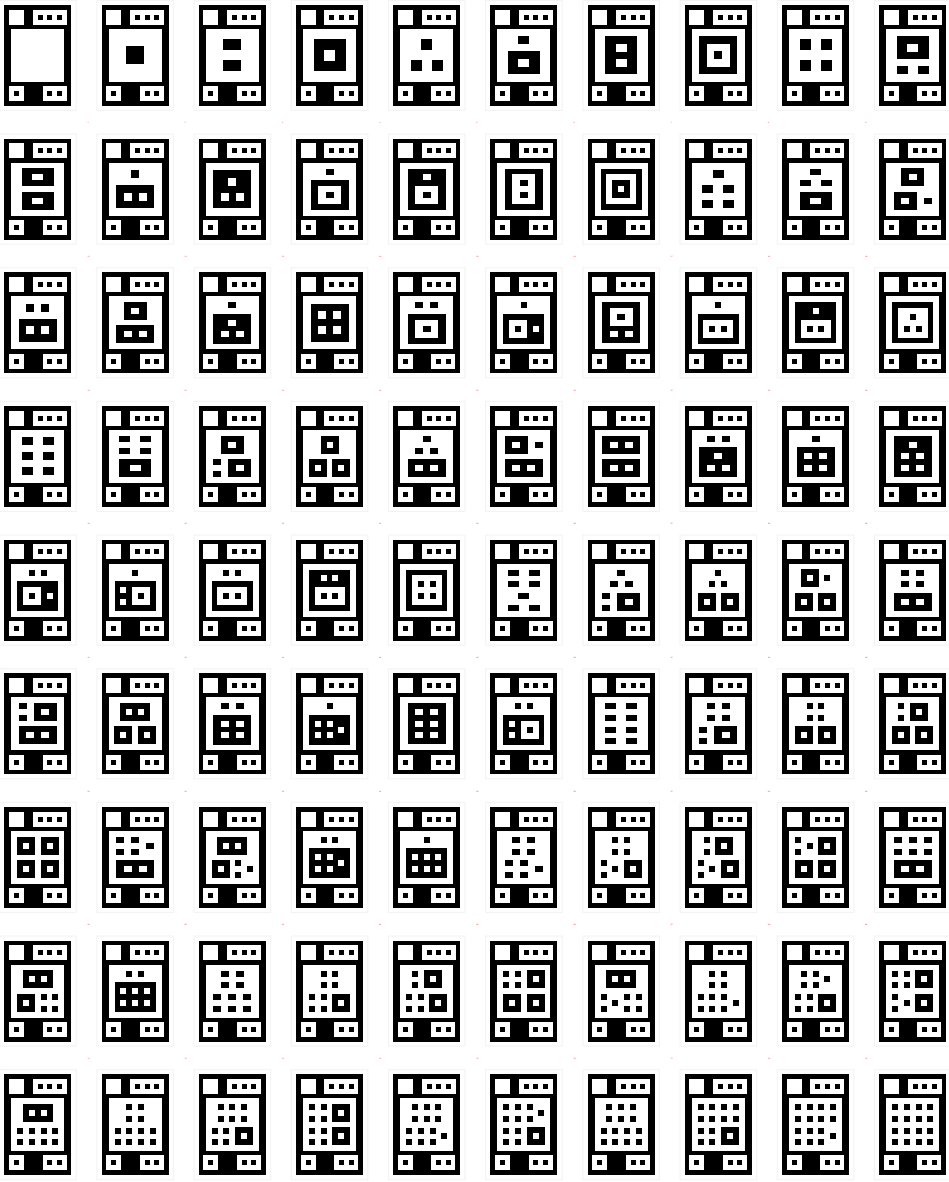

Figure 5.37: Entire set of targets for a 9×9 grid

No of Targets as a Function of Grid Size and Node Count																									
grid size	number of nodes																								
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
4	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
5	1	2	4	5	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
6	1	2	4	5	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
7	1	2	4	8	12	15	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
8	1	2	4	8	12	15	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
9	1	2	4	8	17	30	45	56	65	72	77	81	84	86	88	89	90	90	90	90	90	90	90	90	90
10	1	2	4	8	17	30	45	56	65	72	77	81	84	86	88	89	90	90	90	90	90	90	90	90	90
11	1	2	4	8	17	37	75	130	191	252	306	350	386	411	427	438	444	448	449	449	449	449	449	449	449
12	1	2	4	8	17	37	75	130	191	252	306	350	386	411	427	438	444	448	449	449	449	449	449	449	449
13	1	2	4	8	17	37	85	187	371	639	984	1379	1799	2204	2554	2824	3016	3136	3204	3235	3248	3252	3254	3255	3256
14	1	2	4	8	17	37	85	187	371	639	984	1379	1799	2204	2554	2824	3016	3136	3204	3235	3248	3252	3254	3255	3256
15	1	2	4	8	17	37	85	200	469	1022	2000	3525	5662	8389	11564	14941	18254	21269	ℵ	ℵ	ℵ	ℵ	ℵ	ℵ	ℵ
16	1	2	4	8	17	37	85	200	469	1022	2000	3525	5662	8389	11564	14941	18254	21269	ℵ	ℵ	ℵ	ℵ	ℵ	ℵ	ℵ

Table 5.6: Number of targets

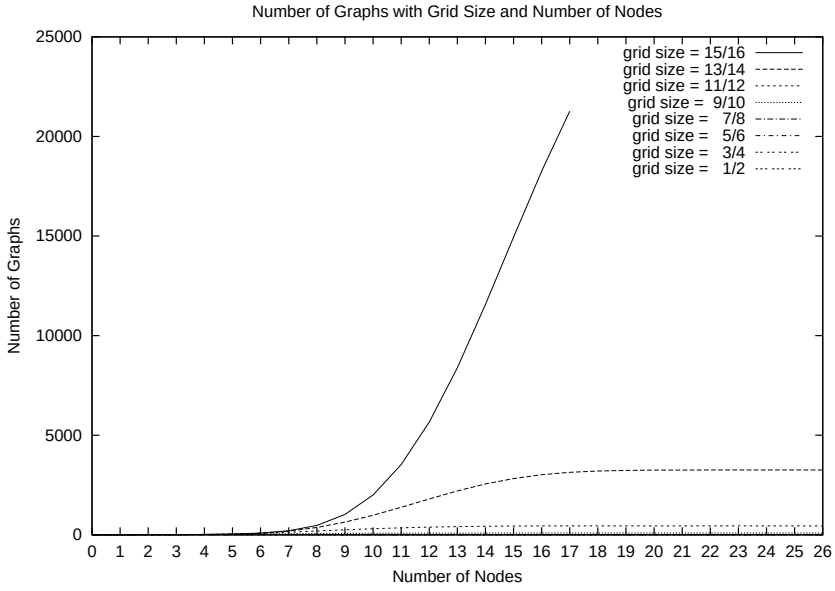

Figure 5.38: Number of graphs as a function of grid size and node count (non-logarithmic plot)

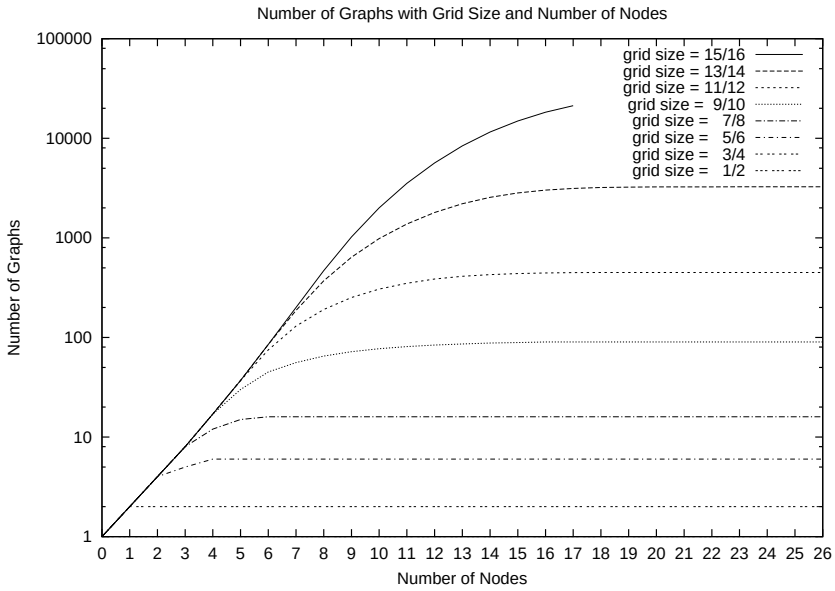

Figure 5.39: Number of graphs as a function of grid size and node count (logarithmic plot)

Maximum No of Targets as a Function of Grid Size	
grid size	no of targets
1	1
2	1
3	2
4	2
5	6
6	6
7	16
8	16
9	90
10	90
11	449
12	449
13	3256
14	3256
15	$>$ 21269
16	$>$ 21269

Table 5.7: Maximum number of targets for each grid size

significance of this trend is encouraging. To move from 449 to well over 21269 targets, requires an increase in linear resolution of only around 30% (*i.e.* a grid size increase $11 \rightarrow 15$). A move to a HDTV camera instead of a PAL camera would more than cover this. The set of 21269+ targets could be used already, provided they were reasonably close to the camera. One could consider an enhanced technique, whereby the reliable depth of the ID RAG tree of a target is computed from its size, and ID extraction is only calculated down to this level.

5.8 Region Processing for RAG Target Detection

The image processing required to form a region decomposition of a scene suitable for target detection using RAGs, is different to that for edge processing. To pick up detail in areas of both bright and dim illumination some form of adaptive filtering is required. Let us calculate the average intensity and average intensity variance within localities, on which to base this adaptive filtering. For simplicity rectangular regions are used.

5.8.1 Adaptive Filtering

Assume that the dimensions of image I are $w \times h$, and the filter (or region) half-widths and half-heights are n_x and n_y. The filter dimensions are odd *i.e.* $(2n_x + 1) \times (2n_y + 1)$. For each pixel in the image the square of the variance in the surrounding region is calculated as:

```
variance2[x,y] =  count * sum2[x,y] - sum[x,y] * sum[x,y];
```

where `count` is the number of pixels in the neighbourhood; `sum[x,y]` represents the sum of the pixel values in the region and `sum2` is the corresponding sum of the squares of the pixel values. Both `sum` and `sum2` are computed for efficiency using a differential or "sliding mean" method described in Appendix F. An upper bound on this expression is:

```
limit = ( std_dev_thresh * std_dev_thresh ) * ( count * count );
```

where `std_dev_thresh` is a user supplied parameter, indicating the number of standard deviations to include when calculating the threshold to decide whether a region is intermediate in intensity. The code to colour a pixel either gray (near average intensity); white (above average intensity) or black (below average intensity) is as follows;

```
        if ( variance2[x,y] < limit )
        {
                    c[x,y] = GRAY;
        }
        else if ( I[x,y] * count > sum[x,y] )
        {
                    c[x,y] = WHITE;
        }
        else
        {
                    c[x,y] = BLACK;
        }
```

An alternative initial conditional, which looks at the variance to intensity ratio rather than the absolute value of the variance is:

```
    (count * 65536.0) / ((std_dev_thresh * std_dev_thresh + 65536.0) * sum2[x,y])
    <
    sum[x,y] * sum[x,y]
```

where the intensity values are known to be in the range 0 to 255. One might expect this to tease out grey values more successfully in areas of low intensity. This was observed to be the case. Overall, though, there was little to chose between the absolute or relative approaches in practice, even though there were visibly observable differences. As a quantitative comparative evaluation was not carried out, the absolute approach was arbitrarily chosen. Figure 5.40 shows the image of a RAG target as captured by a video camera. When a sufficiently small range of near-average intensities is painted gray, this is equivalent to simply converting the image into a two-level black and white image as in Figure 5.41. It can be observed that near edge transitions, black and white regions of the target have been correctly identified. However, when the filter fits totally into a region of constant colour salt and pepper noise appears, as relatively small intensity variations falsely appear significant. Now while the use of a larger filter would circumvent this problem, the very size of filter required would lose fine grained adaptability to intensity variation. Also there is an upper limit on the size of filter required to solve this problem - very large filters both increase computation and complicate the software engineering. The additional categorisation of gray regions is a much more effective solution to the problem. The tri-level image in Figure 5.42 is very clean and the black and white regions correctly reflect those in the target. Extracting information from this tri-level image is addressed in Section 5.8.4.

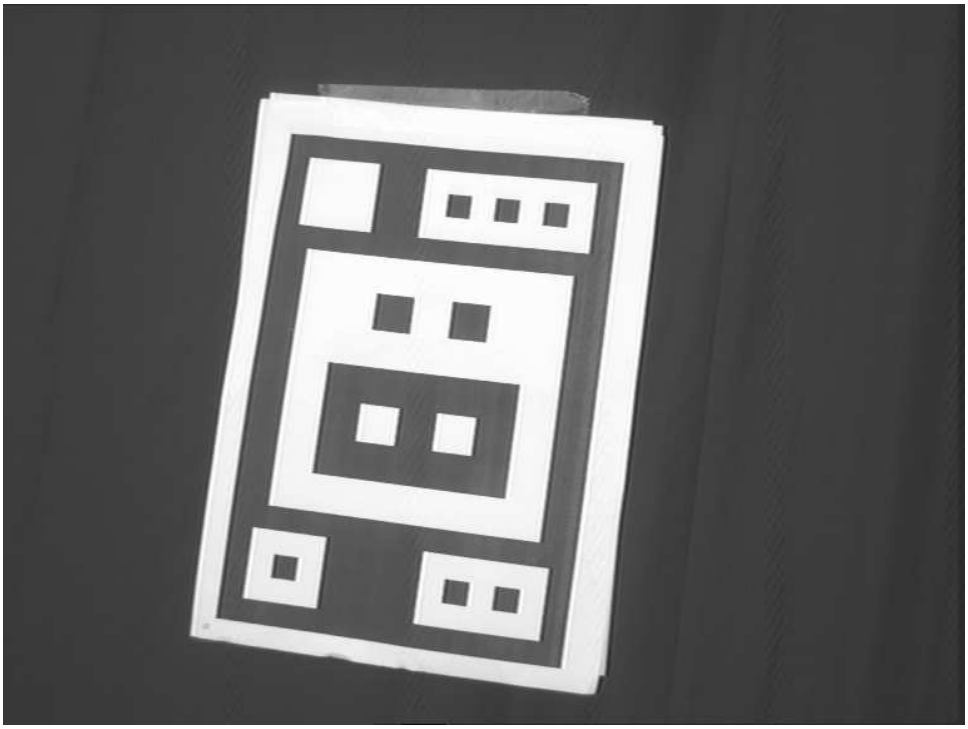

Figure 5.40: Source image

5.8.2 Boundary Conditions

An average image cannot be strictly filtered round the edge as a full neighbourhood is not present. Different approaches are possible:

1. omit processing near edges
2. use wraparound values
3. clamp values “outside” boundary to those of boundary
4. use “mirror” boundary conditions

It was decided to make the average image the same size as original image for simplicity of approach but to avoid 1 which could introduce strange boundary artifacts at the discontinuity. 2 would mix-up the right- and left-hand sides of the image (bad for locally adaptive filters), and clamping the image 3 could potentially introduce artifacts as a set of constant values is hardly natural. In the end, the mirror boundary condition was used as it was expected to give a “natural” result. The original image is increased in size before processing and the new pixels are set to reflected values as in Figure 5.43.

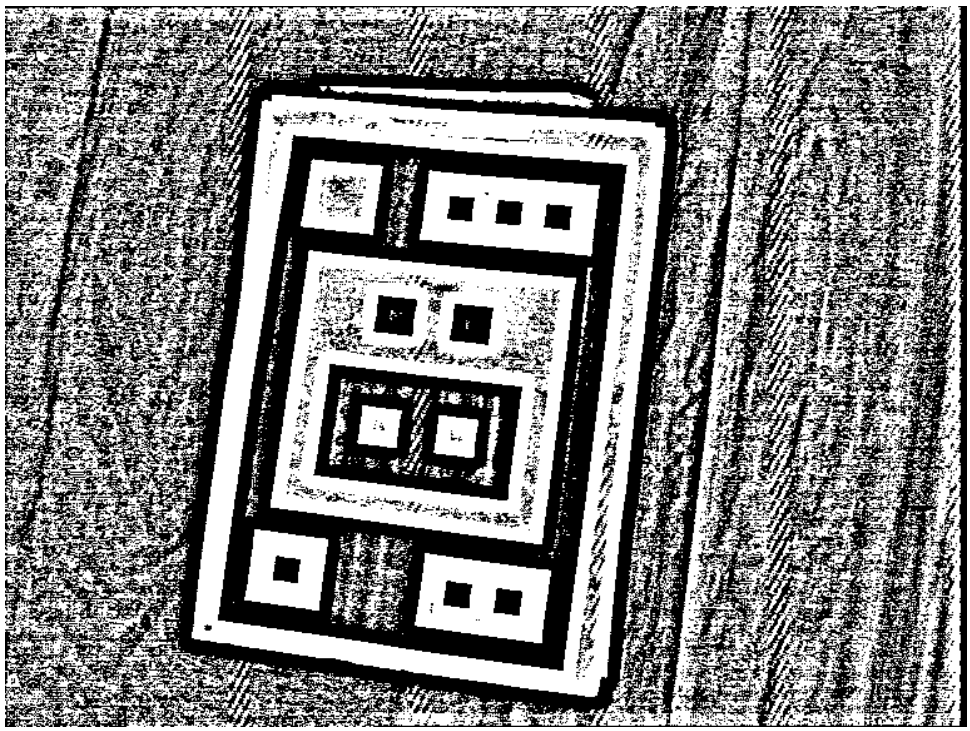

Figure 5.41: Bi-level processed image

5.8.3 Region Creation

Each of the spatially distinct delineated gray, black and white regions is given a unique integer ID or "colour". An image of coloured regions is thus produced, where the value of each pixel is the number of the region in which it finds itself. The algorithm to achieve this is textbook material, and can be summarised as follows:

(A) Scan the image left to right, top to bottom. For each pixel

1. if only one of the upper and left neighbours has the same intensity then copy its colour
2. if both the upper and left neighbours have the same intensity and colour then copy the colour
3. if both the upper and left neighbours have the same intensity but different colours, place the colours in an equivalence relationship, copy the colour with the higher numeric value
4. otherwise, assign a new colour and enter into a new equivalence set.

(B) Replace each pixel's colour by the lowest colour in its equivalence set.

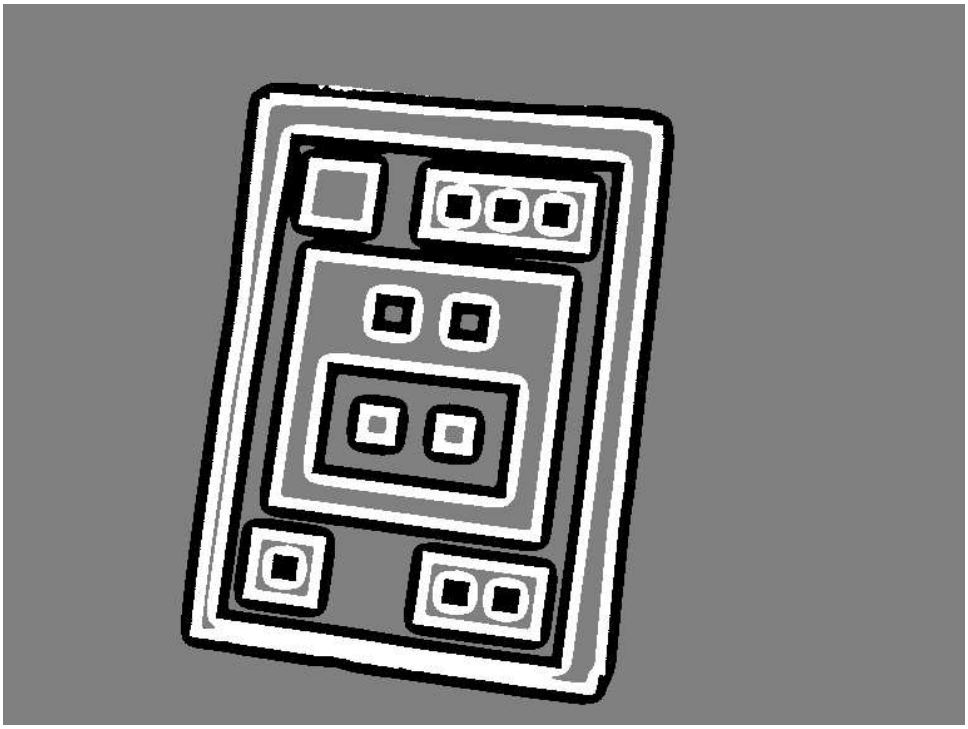

Figure 5.42: Tri-level processed image

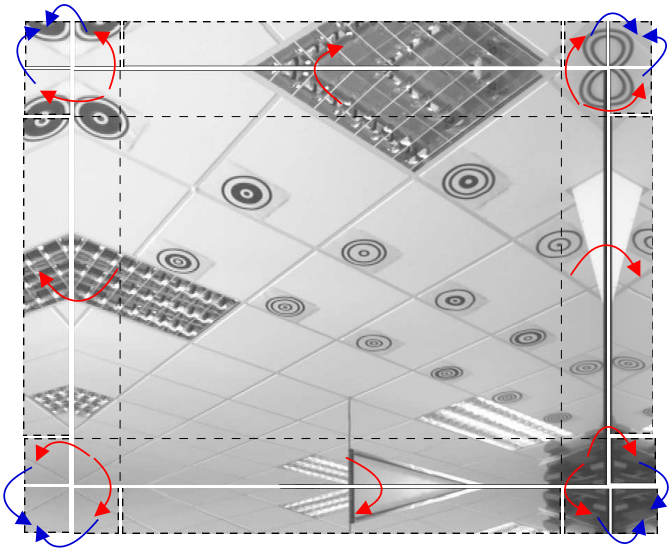

Figure 5.43: Mirroring to handle image boundaries

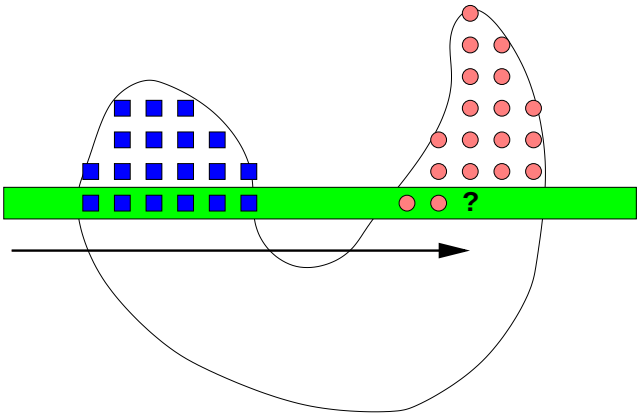

Figure 5.44: Creating two distinct regions

The equivalence sets are necessary, because different parts of the same region could be started to be painted in two different colours (Figure 5.44), before the connection becomes apparent (Figure 5.45). Step (B) involves a global sort for each equivalence set. An algorithm was developed which performs the sort incrementally (and hence more efficiently). In fact, the algorithm developed is array based rather than set based, so is ideal for implementing in a language such as 'C'. The crux of the algorithm is an array called $repaint$, which stores the equivalence relationships.

$$a \sim b, \quad a \leq b \Leftrightarrow \exists n \in \mathbb{W} \quad \text{such that} \quad repaint^n[b] = a$$

where we recursively define $repaint^n[a]$ as $repaint[repaint^{n-1}[a]]$ and $repaint^0[a]$ as a. In particular:

$$repaint[b] = a \Rightarrow a \sim b, \quad a \leq b$$

It is guaranteed that $a \leq b$, because an array entry is initially set up as $repaint[a] = a \quad \forall a \in \mathbb{W}$ and its value can only subsequently set to something lower. By following the implied downward chain in $repaint$, we ultimately reach the smallest (current) member e of the equivalence set E.

$$e \equiv repaint[repaint[\cdots repaint[a] \cdots]]$$

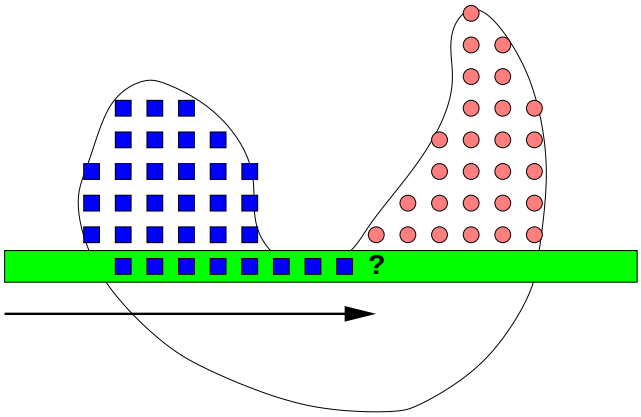

Figure 5.45: Region merging is necessary

In strict mathematical language:

$$e \equiv repaint^n[a] \quad \text{where} \quad \forall m > n \quad repaint^n[a] = repaint^m[a] \quad \forall a \in E, \quad m, n \in \mathbb{W}$$

Note that algorithmically we follow this downward chain until we find the value e, such that $e = repaint[e]$. The textbook algorithm is modified, so that when the intensities are the same but the colours are different, the new colour is dynamically chosen as the lower of the two lowest members of the respective equivalence sets. The equivalence classes for each colour are merged by linking the two lowest element e_{lower} and e_{upper} , where $e_{lower} < e_{upper}$, thus:

$$repaint[e_{upper}] = e_{lower}$$

Note that if the linkage was merely made at the level of the two directly encountered colours, that colours below this level could still point to the previous lowest member of the equivalence class. All elements of both downward chains are then set to e_{lower}, to speed subsequent descents *i.e.*

$$\begin{aligned} repaint[a] &= e_{lower} \\ repaint[repaint[a]] &= e_{lower} \\ \cdots &= e_{lower} \end{aligned}$$

This is not an essential step, but one done from efficiency. Note that even after this step, not all colours will point directly to the smallest colour within their equivalence set, because we could have encountered a colour in the middle of a descent chain and we only set-up direct linking below this point. It is difficult to formally reason about the equivalence relation representation because it is not guaranteed to be maximally directly linked (though it is likely to be near to this condition). If circumstances were other, it would be nice to prove that the algorithm preserves the quality of maximal direct linkedness, which is defined:

$$a \sim b \Leftrightarrow repaint[b] = repaint[a]$$

It is even difficult to ascertain whether the corresponding code which was written for efficiency, adds more overhead than it saves! In practice, however, the optimised code represents an extremely infrequent condition, and there is unlikely to be any measurable overhead. Note that for efficiency of addressing, `pa`, `pa_left` and `pa_up` all reference the same array in memory, but are appropriately offset. The benefit is that the same array index can be used for all. The code for each pixel is shown in Figure 5.46. A final pass is necessary to repaint all regions as the lowest colour in their equivalence set. However, the lowest equivalence set colours are not contiguous in the range $1 \ldots n$ (where n is the number of distinct regions) which is the ideal scenario. To remedy this we scan through `repaint`.

```
ncols = 1;
for(i=0;i<=colour_index;++i)
{
        if ( repaint[i] == i  )
        {
                /* lowest colour in equivalence set - re-number sequentially */
                repaint[i] = ncols;
                ncols++;
        }
        else
        {
                /* set to lower down equivalent colour */
                repaint[i] = repaint[repaint[i]]; /* line DD */
        }
}
```

As we are scanning from the bottom of the `repaint` array upwards, we can be guaranteed that each link generated by line DD, is maximally direct. Note that as we are assured maximal directness at

```
/*      input:  3-level intensity image -  pa
 *      output: coloured region image   -  ca
 *      index is 1D index into 2D image array
 */
if (pa[index] != pa_up[index])
{
        if (pa[index] != pa_left[index])           /* neither same as left nor up */
        {
                colour_index++;                    /* new colour */
                repaint[colour_index] = colour_index;
                ca[index] = colour_index;
        }
        else                                       /* same as left only */
        {
                ca[index] = ca_left[index];        /* propagate colour */
        }
}
else
{
        if (pa[index] != pa_left[index])           /* same as up only */
        {
                ca[index]= ca_up[index];           /* propagate colour */
        }
        else
        {
                if (ca_up[index] == ca_left[index]) /* left & up colours same */
                {
                   ca[index] = ca_up[index];       /* propagate colour */
                }
                else                               /* conflicting colours */
                {
                        /* find lowest member of left equivalent set */
                        small_left = ca_left[index];
                        while ( small_left != repaint[ small_left ] )
                        {
                                small_left = repaint[ small_left];
                        }

                        /* find lowest member of upper equivalent set */
                        small_up = ca_up[index];
                        while (   small_up != repaint[ small_up   ] )
                        {
                                small_up   = repaint[ small_up   ];
                        }

                        /* set up equivalence relation between smallest members */
                        upper = MAX(small_up, small_left);
                        lower = MIN(small_up, small_left);
                        repaint[upper] = lower;
                             ca[index] = lower;

                        /* shorten linkages for efficiency - START */
                        /* set all equivalence chains below to point to lower */
                        small = ca_left[index];
                        while ( small != repaint[ small ] )
                        {
                                old_small = small;
                                small = repaint[ small];
                                repaint[ old_small ] = lower;
                        }

                        small = ca_up[index];
                        while ( small != repaint[ small ] )
                        {
                                old_small = small;
                                small = repaint[ small];
                                repaint[ old_small ] = lower;
                        }
                        /* shorten linkages for efficiency STOP */
                }
        }
}
```

Figure 5.46: Per-pixel region painting code

each stage only the expression:

$$repaint[repaint[i]] \quad \text{is required}$$

$$\text{instead of} \quad repaint[repaint[\cdots[repaint[i]\cdots]] \quad \text{to the appropriate depth}$$

which would be necessary if we were trying to find how to repaint an individual region. Given maximal directness and sequential colour numbering the picture (which is being treated as a linear array) can be repainted.

```
        while(nbytes--)
        {
                ca[nbytes] = repaint[ca[nbytes]];
        }
```

The result of the time-efficient representations for equivalence classes, is to make region generation cheap, whereas it could potentially be a processing bottleneck. Now that the regions are correctly coloured, statistics can be collected on each region. Regions below a certain pixel count can be painted to 0, which will remove both noise and fine detail. The pixel count threshold is a user supplied parameter. While a pixel count threshold proved useful in edge processing, distant RAG targets were successfully detected even though a region only consisted of a single pixel. In practice, the pixel threshold was set to around 5 in edge processing, and 0 in region processing.

5.8.4 Graph Creation

The algorithm to turn an image with regions into a RAG representation g is for each pixel:

```
/* pixel has been cleared to black (i.e. 0) and so deemed of no interest */
if ( ca[index] == 0 )
{
        continue; /* to next pixel */
}
/* different colour from non-black pixel to left */
if  (  (ca_left[index] != 0 ) && (ca[index] != ca_left[index]) )
{
        /* connects two colours in graph g */
        add_arc( g, ca_left[index],ca[index]);
}
/* different colour from non-black pixel above  */
if ( (ca_up[index] != 0) && ( ca[index] != ca_up[index] ) )
{
        /* connects two colours in graph g */
        add_arc( g, ca_up[index], ca[index]);
}
```

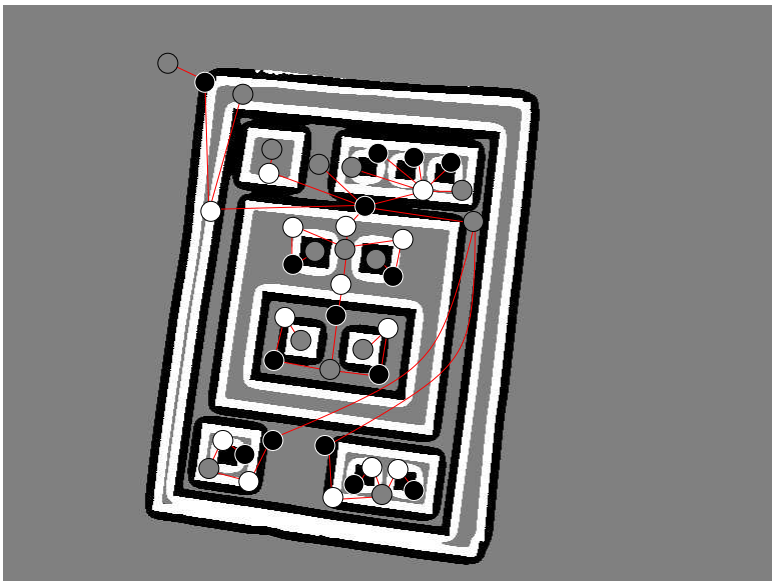

Figure 5.47: RAG superimposed on processed image

This code has the same structure as that for identifying regions, and initial versions of the software merged the two. However, there is little efficiency loss and much gain of clarity by separating the two stages. The above code will potentially add the same arc a number of times: this is equivalent to a single instance of the arc. Figure 5.47 superimposes the generated RAG on top of the generating image. Note that the RAG nodes are placed so that they overlap at least to some extent the region to which they correspond. By careful manual placement it proved possible to create a graph with no intersecting arcs - though this will probably not be achievable in the general case. Figure 5.48 shows the generated RAG in isolation from the visually confusing background image, though still in image space coordinates. Figure 5.49 again shows the same scene RAG, but this time with a rationalised layout, that clearly reveals the portions of the RAG that derive from ID and key portions of the target. There is atypically little extra-target RAG information in this scene RAG, but this image was specially chosen to make the illustrative conversion of an image to RAG tractable. Even so, some small regions in the original image have been ignored for convenience: they make no difference to the line of argument.

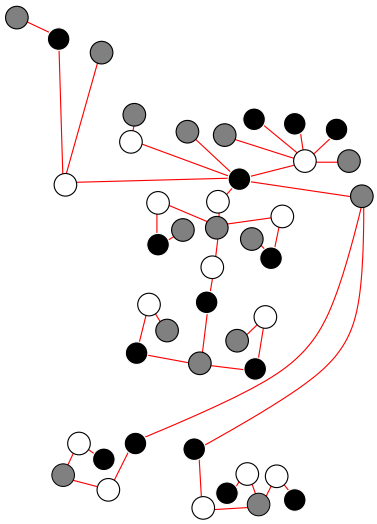

Figure 5.48: RAG in image space coordinates

5.8.5 Slimming Graph

The scene has been converted into a RAG of white, gray and black nodes. Each separate "coloured" region is tagged with its original colour in the three-level image. Gray nodes have been detected as they are a way of picking up regions of indeterminate colour *i.e.* neither clearly black nor white). Throwing away these gray regions gives considerable immunity to noise. However, we cannot simply remove these gray regions from the scene graph. Consider a white region, surrounding a gray region, surrounding a white region which is in turn surrounding a black region in the processed scene as in the bottom left of Figure 5.42. In the real world the white regions are connected. Whereas the adaptive filter, being smaller than the white region, has given significance to the small intensity variations found in the middle of the white region. When we remove the gray region therefore we have to connect up the two white regions in a merging operation. What form should this merging operation take?

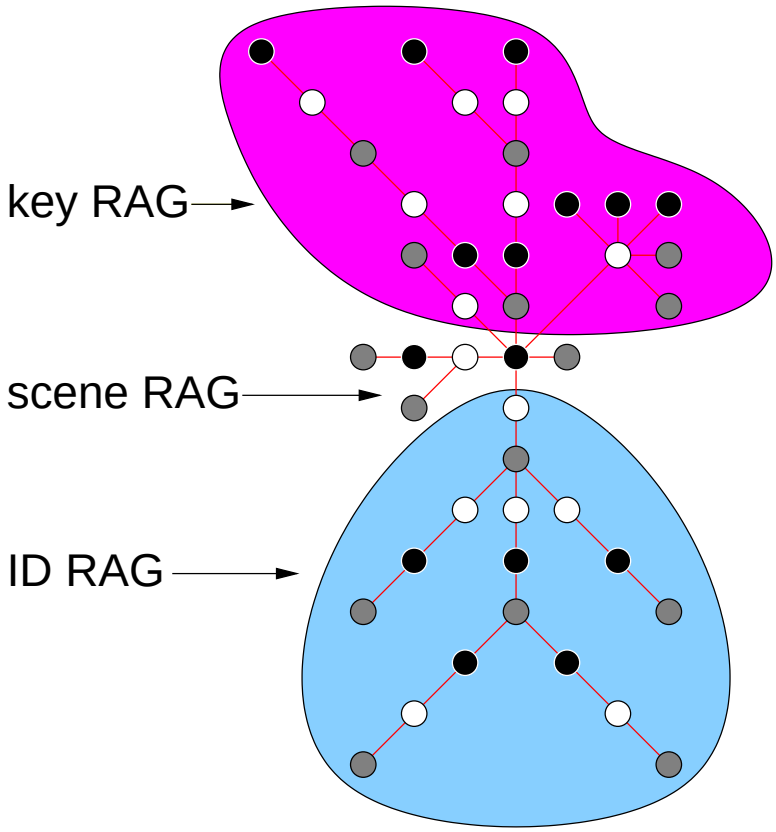

Figure 5.49: RAG with rationalised layout

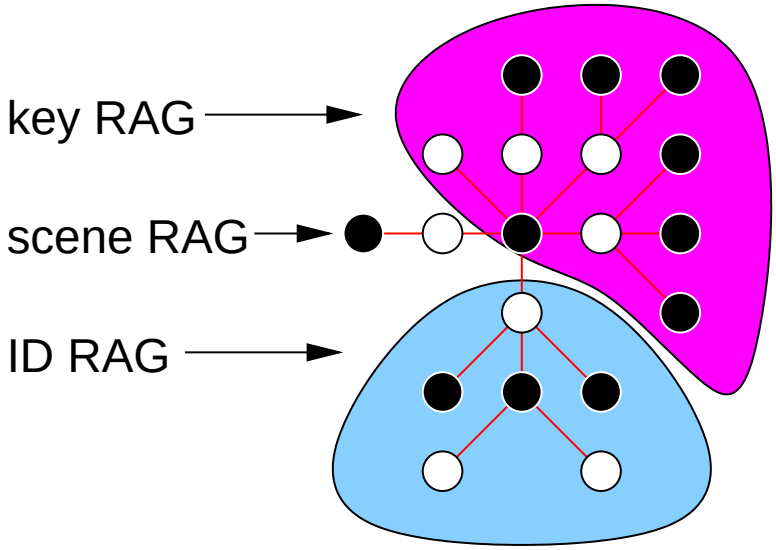

Figure 5.50: RAG with rationalised layout — grey nodes removed

Proposed Solution 1

1. connect all neighbours of each grey area to each other (to restore connectivity)
2. remove the gray areas from the RAG

This does not work. As with the previously described scenario, the single white region, now appears as many connected white regions in the RAG, and so fails to match the target. Valid connectivity patterns in the RAG would have to be re-established.

Proposed Solution 2

1. determine dominant (by region count) colour of neighbours of each grey region
2. paint all the neighbours this dominant colour
3. merge neighbours into a single region
4. remove the gray areas from the RAG

After a possible repainting of the regions, valid connectivity would again have to be established.

Proposed Solution 3

1. merge all white neighbours of each grey area with each other
2. merge all black neighbours of each grey area with each other
3. remove the gray areas from the RAG

This schema advantageously allows the connectivity already established in the existing RAG to stay valid, rather than having to re-establish it as in solutions 1 and 2. Eventually, after considerable thought, it becomes clear that all adjoining white regions have to be merged and all adjoining black regions have to be merged. In fact, no merging operation other than solution 3 is possible. Adjacency relationships are, however, being added which may not exist in the real world or which may not be valid topologically *i.e.* we may be departing from a form which is realisable as an image. However, the significance is that the real world target RAGs that we require should be embedded in these new adjacency relationships. This removal of grey nodes is a pure graph processing operation, and is

called “graph slimming” as it removes surplus gray nodes to reveal the core black and white adjacency relationships. Dealing with regions of ambiguous intensity in graph space is much more efficient than doing so in image space.

5.8.6 Graph Searching

To locate the RAG targets within a scene, the graph of the key RAG common to all targets is searched for within the RAG representation of the scene. Typically, multiple instances of this key RAG will be found. The problem of target location is reduced to one of graph searching. Although the target RAGs are trees, they are being located by more generic graph searching, because the scene RAG is most likely a true graph (*i.e.* it may contain loops).

A graph in the ML implementation is simply a collection of nodes, and a collection of arcs connecting these nodes. The nodes are represented as integers. Note that a colour (*i.e.* black, white or gray) is not associated with each node, which is the case in practice. However, for the initial development of the ML code, this aspect was ignored. Finding the RAG key graph is sufficient to locate the target, for extra security the colours of the corresponding nodes can be checked for equality. Using colour will also allow the earlier rejection of unsuitable matches, enhancing performance. The datatype definitions in ML are:

```
type Node  = int;
type Nodes = Node list;
type Arc   = Node * Node;
type Arcs  = Arc list;
type Graph = Nodes * Arcs;
```

A typical scene graph is shown in Fig 5.51. How was this graph obtained? The ‘C’ code which processed the scene into a set of connected regions, also printed out an ML representation of the RAG. This RAG was then used from within ML to prototype the graph searching algorithm. As six targets were visible in the scene: finding six embedded key RAGS was verification of the graph searching algorithm’s correctness.

The graph for the key RAG is:

```
val s = ( [ 1, 2, 3, 4, 5, ~6, ~7, 8, 9, 10, 11, 12, 13 ],
          [
                (1,2), (1,3), (1,4), (1,5), (1,~6), (1, ~7) ,
                (3,8), (4,9), (4,10), (5,11), (5,12), (5,13)
          ]
        ): Graph;
```

```
val g = ( [
1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
61, 62, 63, 64, 65, 66, 67, 68, 69, 70,
71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
81, 82, 83, 84, 85, 86, 87, 88, 89, 90,
91, 92, 93, 94, 95, 96, 97, 98, 99, 100,
101, 102, 103, 104, 105, 106, 107, 108, 109, 110,
111, 112, 113, 114, 115, 116, 117, 118, 119, 120,
121, 122, 123, 124, 125, 126, 127, 128, 129, 130,
131, 132, 133, 134, 135, 136, 137, 138, 139, 140,
141, 142, 143, 144, 145, 146, 147, 148, 149, 150,
151, 152, 153, 154, 155, 156, 157, 158, 159, 160,
161, 162, 163, 164, 165, 166, 167, 168, 169, 170,
171, 172, 173, 174, 175, 176, 177, 178, 179, 180,
181, 182, 183, 184, 185, 186, 187, 188, 189, 190,
191, 192, 193, 194, 195, 196, 197, 198, 199, 200,
201, 202, 203, 204, 205, 206, 207, 208, 209, 210,
211, 212, 213, 214, 215, 216, 217, 218, 219, 220,
221, 222, 223, 224, 225, 226, 227, 228, 229, 230,
231, 232, 233, 234, 235, 236, 237, 238, 239, 240,
241, 242, 243, 244, 245, 246, 247, 248, 249, 250,
251, 252, 253, 254, 255, 256, 257, 258, 259, 260,
261, 262, 263, 264, 265, 266, 267, 268, 269, 270,
271, 272, 273, 274, 275, 276, 277, 278, 279, 280,
281, 282, 283, 284, 285, 286, 287, 288, 289, 290,
291, 292, 293, 294, 295, 296, 297, 298, 299, 300,
301, 302, 303, 304, 305, 306, 307, 308, 309, 310,
311, 312, 313, 314, 315, 316, 317, 318, 319, 320,
321, 322, 323, 324, 325, 326, 327, 328, 329, 330,
331, 332, 333, 334, 335, 336, 337, 338, 339, 340,
341, 342, 343, 344, 345, 346, 347, 348, 349, 350,
351, 352, 353, 354, 355, 356, 357, 358, 359, 360,
361, 362, 363, 364, 365, 366, 367, 368, 369, 370,
371, 372, 373, 374, 375, 376, 377, 378, 379, 380,
381, 382, 383, 384, 385, 386, 387, 388, 389, 390,
391, 392, 393, 394, 395, 396, 397, 398, 399, 400,
401, 402, 403, 404, 405, 406, 407, 408, 409, 410,
411, 412, 413, 414, 415, 416, 417, 418, 419, 420,
421, 422, 423, 424, 425, 426 ],
[   (1,2), (4,5), (5,6), (4,7), (7,8), (3,9), (9,10), (9,11), (1,17), (17,18), (2,18)
, (3,20), (4,21), (4,23), (20,24), (4,25), (3,26), (9,29), (9,31), (17,32), (3,33)
, (9,34), (3,35), (3,36), (35,38), (3,40), (36,42), (36,43), (37,44), (37,45), (43,46)
, (43,47), (43,48), (41,49), (44,50), (45,51), (45,52), (41,53), (37,54), (49,55), (52,56)
, (52,57), (52,58), (49,59), (53,60), (38,61), (55,62), (55,63), (55,64), (36,65), (50,66)
, (65,67), (41,68), (45,69), (49,70), (67,71), (69,72), (70,73), (60,74), (41,75), (72,76)
, (3,77), (65,78), (70,79), (70,80), (78,81), (65,82), (69,83), (60,84), (41,85), (70,86)
, (70,87), (83,88), (9,89), (86,90), (70,92), (36,93), (36,94), (94,95), (93,96), (93,97)
, (45,98), (45,99), (49,100), (99,101), (3,102), (49,103), (98,104), (99,105), (100,107), (100,108)
, (103,109), (37,110), (9,111), (12,116), (9,117), (9,119), (9,120), (9,121), (9,122), (9,123)
, (3,124), (9,125), (3,126), (3,127), (3,128), (125,130), (9,131), (125,132), (9,133), (13,129)
, (129,134), (3,135), (9,136), (9,137), (136,138), (9,139), (9,140), (138,141), (9,141), (9,142)
, (3,143), (129,144), (3,145), (3,146), (3,147), (3,148), (124,149), (149,150), (3,150), (117,151)
, (9,152), (117,153), (147,154), (9,155), (9,156), (117,157), (151,158), (3,159), (153,160), (154,161)
, (153,162), (9,163), (158,164), (9,165), (160,166), (160,167), (160,168), (152,169), (152,170), (9,171)
, (3,172), (169,173), (117,174), (170,175), (3,176), (170,177), (164,178), (161,179), (3,180), (171,181)
, (171,182), (173,183), (3,184), (3,185), (175,186), (175,187), (175,188), (9,189), (171,190), (153,191)
, (180,192), (9,193), (176,194), (171,195), (191,196), (170,197), (9,198), (3,199), (191,200), (9,201)
, (191,202), (197,203), (14,204), (203,205), (124,206), (191,207), (152,208), (152,209), (9,210), (3,211)
, (207,212), (9,213), (9,214), (197,215), (197,216), (9,217), (124,218), (124,219), (9,220), (152,221)
, (9,222), (153,223), (218,224), (219,225), (219,226), (9,227), (153,228), (9,229), (223,230), (9,231)
, (223,232), (228,233), (3,234), (9,235), (152,236), (231,237), (227,237), (170,238), (170,239), (15,240)
, (238,241), (238,242), (239,243), (9,244), (9,245), (9,246), (227,247), (9,248), (152,249), (152,250)
, (152,251), (3,252), (152,253), (253,254), (9,255), (9,256), (152,257), (9,258), (152,259), (152,260)
, (9,261), (9,262), (152,263), (152,264), (9,265), (152,266), (9,267), (152,268), (227,269), (152,270)
, (3,271), (152,272), (152,273), (9,274), (152,275), (152,276), (152,277), (9,278), (152,279), (152,280)
, (9,281), (9,282), (9,283), (152,284), (152,285), (3,286), (152,287), (227,288), (9,289), (152,290)
, (9,291), (227,292), (152,293), (152,294), (9,295), (9,296), (9,297), (152,298), (9,299), (9,300)
, (152,301), (152,302), (9,303), (3,304), (3,305), (3,306), (3,307), (9,308), (9,309), (9,310)
, (3,311), (306,312), (307,313), (307,314), (308,315), (309,316), (313,317), (313,318), (313,319), (309,320)
, (3,321), (9,322), (315,323), (312,324), (320,325), (316,326), (315,327), (316,328), (309,329), (323,330)
, (323,331), (323,332), (326,333), (326,334), (326,335), (322,336), (324,337), (307,338), (3,339), (325,340)
, (336,341), (336,342), (338,343), (315,344), (316,345), (3,346), (345,347), (345,348), (344,349), (338,350)
, (345,351), (345,352), (349,353), (350,354), (3,355), (345,356), (345,357), (345,358), (9,359), (307,360)
, (307,361), (9,362), (9,363), (9,364), (360,365), (360,366), (361,367), (3,368), (9,369), (315,370)
, (316,371), (9,372), (315,373), (316,374), (9,375), (370,376), (370,377), (371,378), (371,379), (373,380)
, (9,381), (374,382), (9,383), (9,384), (9,385), (9,386), (9,387), (9,388), (9,389), (9,390)
, (3,391), (9,392), (9,393), (394,395), (9,396), (372,397), (3,398), (372,399), (372,400), (9,401)
, (3,402), (398,403), (3,404), (9,405), (9,406), (3,407), (9,408), (9,409), (9,410), (9,411)
, (9,412), (9,414), (9,415), (9,416), (9,413), (9,417), (9,418), (411,419), (411,420), (410,421)
, (412,422), (411,423), (418,424), (422,425)]) : Graph;
```

Figure 5.51: Textual representation of scene graph

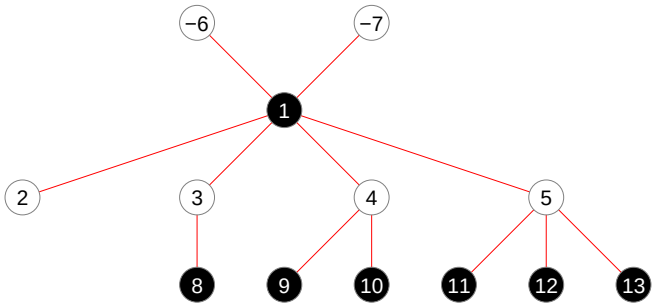

Figure 5.52: Key RAG

Note that the unary minus operator in ML is represented as ~ to distinguish it from the binary subtraction operator. Figure 5.52 gives an equivalent graphical representation of the key RAG. Nodes with negative integer representations serve a rather special function. These indicate the regions which connect the key design with the real world on the outside and the ID design in the inside. It is at these connection points, that graph searching should be truncated. A negative node value therefore acts as a halt signal to the matching algorithm. There are no negative nodes in the scene RAG. It could be argued that negative nodes are not necessary and nodes -6 and -7 could simply be white nodes 6 and 7 representing the immediate exterior region and immediate central interior region of the top-level black region. As the targets are merely bi-level then this approach would be correct. However, in the general case the colours of the terminal regions cannot be assumed to be white, and the negative node number acts as a colour wild-card. In essence this is a step towards the idea of a RAG target regular expression, expanded upon in Section 5.7.

The top level function `find_subgraphs` takes the scene graph and the graph to be located as arguments. It returns a list of scene nodes where matches have been found with the top level node (with value 1) of the key RAG.

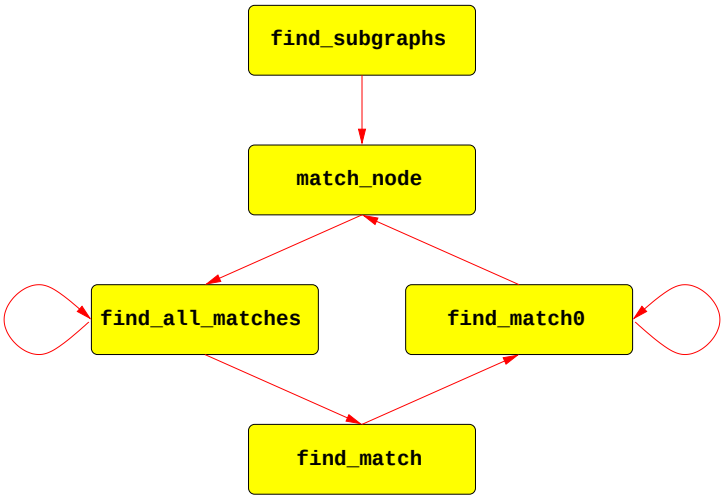

Figure 5.53: Function call structure for graph searching

```
(* find_subgraphs : Graph -> Graph -> Node list *)
fun find_subgraphs  (s:Graph) ((n,a):Graph) =
         filter
         ( match_node s ((n,a):Graph) 1 )
         n;
```

The definition is written in terms of a function `match_node` which establishes whether there is a match between two graphs by aligning named nodes in each. The code for `match_node` is rather complex as it is forms part of a set of four mutually recursive functions, necessarily defined simultaneously in ML using the `and` constructor. The call structure is shown in Fig 5.53. Of the four mutually recursive functions, two are themselves individually recursive. The code in Figure 5.54 is the graph searching algorithm at its most compact. The ML code was ported to 'C' and incorporated within the image processing software. Prototyping the graph searching code in ML facilitated software development. *Ab initio* development of graph searching code in 'C', would have been difficult. However, before the port was undertaken the algorithm was examined for potential increases in efficiency for four reasons:

1. the graph searching is core to target recognition — it should run as quickly as possible
2. changing the code once committed to 'C' would be more difficult

```
(* match_node: Graph -> Graph -> Node -> Node -> bool
   ------------------------------------------------
Do the graphs match when we align the the given nodes?

         s          - key graph         (graph being searched for)
         g          - scene graph       (graph that is searched  )
         s_n        - key node
         g_n        - scene node
 *)
fun match_node ( s : Graph )  ( g : Graph ) s_n g_n =
if ( s_n < 0 )
then    true                                    (* reached end of key graph so match is successful *)
else    let
                val ns = neighbours s s_n;      (* neighbours of node s_n in graph s *)
                val ng = neighbours g g_n;      (* neighbours of node g_n in graph g *)
        in
                if ( length ns <> length ng )
                then    false                   (* no. of neighbours different -> graphs do not match *)
                else    let
                                val g'  = remove_node g g_n;
                                val s'  = remove_node s s_n;
node_pairs (ns, ng)
                        in      (* remove matching nodes and look at
                                   neighbouring nodes from both graphs *)
                                find_all_matches s' g' ns ng
                        end
        end
(* find_all_matches :  Graph -> Graph -> Node list -> Node list -> bool
   -------------------------------------------------------------------
Do the graphs match for some permutation of nodes in key and scene graphs?

         s            - key graph           (graph being searched for)
         g            - scene graph         (graph that is searched  )
         (s_n::s_l)   - key nodes
         g_l          - scene nodes
*)
and find_all_matches ( s : Graph )  ( g : Graph ) []  g_l        = true
        (* no more key nodes to match against -> successful match     *)
 |  find_all_matches ( s : Graph )  ( g : Graph ) (s_n::s_l) g_l =
let
        (* look for match to key node s_n in scene node list g_l      *)
        (* g_l' will be g_l with matching node removed if match found *)

        val ( bool, g_l' ) = find_match s g s_n g_l;
in
        if bool
        then    find_all_matches s g s_l g_l'     (* recurse with rest of scene nodes to match *)
        else    false                             (* no match found -> no overall match *)
end

(* find_match: Graph -> Graph -> Node -> Node list -> bool * Node list
   ------------------------------------------------------------------
interface to recursive function find_match0 which takes an additional
accumulation parameter - initialise this to the null list.
*)
and find_match ( s : Graph ) ( g : Graph ) s_n g_l
          = find_match0 ( s : Graph )  ( g : Graph ) s_n g_l []

(* find_match0: Graph -> Graph -> Node -> Node list -> Node list -> bool*Node list
   -------------------------------------------------------------------------------
Can we match graphs using key node s_n and a scene node from list g_l ?

         s          - key graph            (graph being searched for)
         g          - scene  graph         (graph that is searched  )
         s_n        - key node
         (g_n::g_l) - scene nodes
         acc        - accumulating list of non-matching scene nodes
*)
and find_match0 (s : Graph)  (g : Graph) s_n []         acc = (false,[])
 |  find_match0 (s : Graph)  (g : Graph) s_n (g_n::g_l) acc =

        if ( match_node s g s_n g_n )
        then    (true, acc @ g_l)
        else    find_match0 s g s_n g_l (acc @ [g_n]); (* look for s_n IN g_l *)
```

Figure 5.54: Graph searching algorithm

3. efficiency changes are simpler to implement in ML

4. the nature of the efficiency changes can be informed by knowledge of the target language, namely ‘C’, even if those changes are in ML.

It was realised that one of the efficiency hits in ‘C’ and indeed an implementation difficulty would be the graph recursive step, where one graph is repeatedly formed from another. This occurs in the following lines of ML:

```
val g' = remove_node g g_n;  (* g' is graph g with node g_n removed *)
val s' = remove_node s s_n;  (* s' is graph s with node s_n removed *)
```

Is there any way to avoid this operation, because the allocation and deallocation of memory for such a complex structure as a graph will be awkward? Can the same two top-level graphs be kept throughout? The reason a matched node is removed is to prevent going round in circles as chains of nodes (connected by the neighbour relation) are followed. However, as one of the graphs is a tree then matching is intrinsically prevented from following a cycle. The only cycle to prevent is moving to a neighbour and then back again. Provided we can eliminate these 2-cycles then there is no need to recurse on the graph structure itself. To stop going back the way you have come, the original “parent node” is added as an extra parameter to all these functions. The new `match_node'` is shown:

```
(* match_node': Node -> Node -> Graph -> Graph -> Node -> Node -> bool
   ------------------------------------------------------------------
Do the graphs match when we align the the given nodes?

    sp_n       - key parent node
    gp_n       - scene parent node
    s          - key graph         (graph being searched for)
    g          - scene graph       (graph that is searched  )
    s_n        - key node
    g_n        - scene node
 *)
fun match_node' sp_n gp_n ( s : Graph ) ( g : Graph ) s_n g_n =
if ( s_n < 0 )
then
    true
else
    let (* remove parent nodes *)
        val ns  = remove_from_list sp_n (neighbours s s_n);     (* changed line *)
        val ng  = remove_from_list gp_n (neighbours g g_n);     (* changed line *)
    in
        if ( length ns <> length ng )
        then
            false
        else (* use graph parameters s & g unaltered  *)
            find_all_matches' s_n g_n s g ns ng                 (* changed line *)
    end;
```

The initial call to `match_node'` uses parents nodes of value zero:

```
fun find_subgraphs' (s:Graph) ((n,a):Graph) = filter
                                              ( match_node' 0 0 s ((n,a):Graph) 1 )
                                              n;
```

The node number 0 does not appear in any graph. A graph has node numbers 1 to n, where n is the total number of nodes. A terminal node of an ID graph will be represented by the additive inverse of the node number. Note that the only difference in the remainder of the functions are an additional two parameters `sp_n` and `gp_n` which are passed on unaltered.

5.8.7 Conversion to 'C'

A graph is represented in 'C' by an N by N array where N is the number of nodes. If the array element indexed by (i, j) is non-zero then an arc exists between nodes i and j. If the array value is negative then this indicates either the node i or the node j, has a negative node number (i.e i and j are strictly the absolute value of the node numbers). Which is negative is not important, as long as the graph search is arrested at that stage. The array values used are -1, 0, and 1. The colours of the nodes (black, white or gray), are stored in a linear array of length N. Global memory allocation for each graph at start of day is all that is needed for graph searching. All other data structures are 1D lists which can be implemented as arrays of sufficiently large pre-determined size. In short, the ML can be "bent" in a certain direction before a 'C' translation, to ensure that no difficult target implementation problems will arise. With colour in the picture, the equivalent version of `match_node'` becomes

```
fun match_node'' sp_n gp_n ( s : Graph ) ( g : Graph ) s_n g_n =
if ( colour s s_n <> colour g g_n )
then
    false
    (* colour of node s_n in graph s <> colour of node g_n in graph g
       so match fails right away *)
else
    match_node' sp_n gp_n s g s_n g_n;
```

An unexpected observation was that when ML was translated in 'C', the number of lines of code stays almost exactly the same. The agency of ML assists in the production of concepts that lead to compact expression. These concepts can equally well be used in 'C', though the use of 'C' alone would have have led to lengthier code which did not incorporate these concepts. The translation to 'C' of the "bent" ML code to identify the key RAG, was extremely direct. In fact, no requirement was

identified to verify the colours of corresponding regions found because the graph pattern on its own was sufficient to uniquely identify the key patterns.

5.8.8 Extracting Graph ID

Graph information on its own is **not** sufficient to identify the location of a target's ID RAG, even when its corresponding key tree has been found within a scene. To understand this consider what the six connections from the top level node of a key tree represent:

1. the outside world
2. the inside ID pattern
3. a white region containing 0 black blobs
4. a white region containing 1 black blob
5. a white region containing 2 black blobs
6. a white region containing 3 black blobs

The match is performed against all four white regions containing blobs, but not against the outside world or against the inside ID pattern, because these are unknown. There is therefore genuine ambiguity in which of the two unmatched arcs links to the ID pattern and which links to the outside world. Initially, the software would take the unmatched node of smaller area as the ID pattern — clearly 1 is going to be larger than 2. Note the use of auxiliary information about the region which is not graph based. This worked in practice for a long time, until simple targets with ID graphs the same as the graphs for the white rectangles in the corners appeared within the scene. Here, the ID graph could match successfully against part of the key graph, so the detected ID graph would actually appear to be one of the corner regions! The result was a misidentified target.

By sorting the areas of the regions of the 6 top nodes, we can find the top level outside region (the largest) and the top-level ID region (the second largest). This strategy would seem to permit the use of valid ID graphs of forms 3 to 6. However, correctly identifying all four corner regions (for position purposes) becomes impossible. This is because in the pathological case, the outside world could also match against one of the corner regions. In the light of all such possible confusions, the decision was made to banish all RAG targets with an ID pattern the same as a corner region. This only loses 4 targets, but permits disambiguation in the pathological case.

The following argument illustrates how to correctly identify the corner regions necessary for position determination purposes. The key RAG nodes concerned are 2,3,4,5,6,7. In the ideal case 2,3,4 and 5 match the corresponding corner regions; whereas 6 and 7 match the outside world and the inside ID target (with genuine ambiguity as to which is which). Assume the matching scene nodes have been sorted according to area, and thus the following assignments can be made:

```
/*
  the array sorted_key_id[] contains the key node numbers
  2, 3, 4, 5, 6, 7 sorted according to size of their matching
  regions in index positions  2, 3, 4, 5, 6, 7
 */
inside_id  = sorted_key_id[6];  // will be 6 or 7 numerically
outside_id = sorted_key_id[7];  // will be 7 or 6 numerically
                                // (or 2, 3, 4, 5 in pathological cases)
```

The true value of `inside_id` is either 6 or 7, because it cannot be matched against any of the corner regions 2, 3, 4, or 5. However, we cannot control the outside world, and it is possible the outside world could match against any of the corner patterns. To establish the true locations of the corner patterns (for position purposes) the following logic is used:

```
/*
  the array scene_id contains the mapping established through tree matching of
  target key node ID to scene node ID (which we have shown may not be as ideal
  as we would like)

        c        - image holding "coloured" regions
        xc, yc   - arrays to hold coordinates of corner regions
        region   - data structure holding global informations about all regions
 */

for ( daughter_id = 2;  daughter_id <= 5; ++ daughter_id )
{
        /*   testing daughter_id against inside_id
             is pointless as no confusion is possible  */

        if ( daughter_id == outside_id ) /* pathological case */
        {
                /* daughter will be node that inside is not!  */
                if ( inside_id == 6 )
                {
                        daughter_colour = scene_id[7];
                }
                else
                {
                        daughter_colour = scene_id[6];
                }
        }
        else
        {
                daughter_colour = scene_id[daughter_id];
        }
        /*  here we do position determination
            using correct "colour" of corner region */
        get_region_centroid
        (
                c, daughter_colour, &xc[daughter_id], &yc[daughter_id], regions
        );
}
```

The above code is included because it took considerable thought to resolve the confusion. It is believed that no resolution is possible, without banning the first 4 targets patterns that are generated, and a

proof of this would involve generating a particular counterexample. However, rejecting 4 targets is no hardship in practice — philosophically just a further example of selecting the target set to make processing easy. To turn the ID tree that has now been identified into a unique integer ID, it is first turned into a string. For example the tree in Figure 5.29 would be represented as `(()()(())(()))`. The index of this string in a string table is used as the ID. Indeed this string table is generated by more or less the same ML code that generated the ID trees in the first place. A sample string table is given in Figure 5.55. Note that the ML code produces both a PostScript document containing drawings of the targets, and a 'C' include file which contains string representations of these targets, The conversion of a tree to an integer ID by this process is inefficient as it involves a search, but is not slow.

5.9 Circular Targets Revisited

Some thought was applied to the problem of a lack of target diversity when circular targets were being used. Although the profusion of targets generated by the RAG approach bypasses the issue, some of the solution ideas developed were felt to be of more general applicability and are presented here. Also addressed are some miscellaneous ideas on targets: barcodes are compared against the DataGlyph from Xerox PARC and an idea for a "RAG wallpaper" is proposed.

5.9.1 Circumventing Limited Diversity

What happens if combinations of adjacent or close targets (spatially or even temporally in an image sequence) are used instead of just a single target? For simplicity, assume the targets are on a square grid and that some adjacency information is known *e.g.* at least one rook-wise adjacent target is known. The questions are, by using this knowledge:

1. What is the maximum area coverable by using n different targets?

2. What is the optimal layout of these targets?

What about a using a space-filling curve (Sierpinski, Peano, Hilbert?) to layout the targets? Do these guarantee uniqueness of each local area? One could in theory, of course, alternatively exhaustively generate all permutations and combinations of target layouts, and then subtract out those with identical/similar local conditions. A few simple schemes follow:

```
typedef struct entry
{
         int page;
         char * ascii_graph;
} PageEntry;
PageEntry Pages[] =
{
{0,"()"},
{1,"(())"},
{2,"(()())"},
{3,"((()))"},
{4,"()()()()"},
{5,"(())()()"},
{6,"((()))()"},
{7,"(((())))"},
{8,"()()()()()"},
{9,"(())()()()"},
{10,"((()))()()"},
{11,"(())()()()"},
{12,"((()))()()"},
{13,"(())()()()()"},
{14,"((()))()()()"},
{15,"(())()()()()"},
{16,"((()))()()()"},
{17,"()()()()()()()"},
{18,"(())()()()()()"},
{19,"((()))()()()()"},
{20,"(())()()()()()"},
...
{285,"((((()))()()()()()()()))"},
{286,"((((()))()()()()()()()))"},
{287,"((((()))()()()()()()()))"},
{288,"((((()))()()()()()()()))"},
{289,"((((()))()()()()()()()))"},
{290,"((((()))()()()()()()()))"},
{291,"((((()))()()()()()()()))"},
{292,"((((()))()()()()()()()))"},
{293,"((((()))()()()()()()()))"},
{294,"((((()))()()()()()()()))"},
{295,"((((()))()()()()()()()))"},
{296,"((((()))()()()()()()()))"},
{297,"((((()))()()()()()()()))"},
{298,"((((()))()()()()()()()))"},
{299,"((((()))()()()()()()()))"},
{300,"((((()))()()()()()()()))"},
{301,"((((()))()()()()()()()))"},
{302,"((((()))()()()()()()()))"},
{303,"((((()))()()()()()()()))"},
{304,"((((()))()()()()()()()))"},
{305,"((((()))()()()()()()()))"}
};
```

Figure 5.55: Textual representation of target ID graphs

I	Horizontal Difference	Vertical Difference
1	1	7
2	2	14
3	3	21
4	4	28
5	5	35
6	6	42
7	7	49
8	8	56
9	9	$63 \sim 2$
10	10	$70 \sim 9$
11	11	$77 \sim 16$

Table 5.8: Effect of varying increment I on grid labelling

Using an Adjacent Target

Consider using $n = 61$ out of the 64 circular targets produced and a 7×7 grid of tiles. It is significant that 61 is the largest possible prime number of targets, and the square grid is the largest that can be covered by these. Cover the tiles from top-left to bottom right and increment the target ID by 1 each time, starting at 0. Let us call the increment I. Horizontally adjacent cells will differ by 1. Vertically adjacent cells will differ by 7. Now label another set of 7×7 ceiling tiles with an increment of 2 ($I = 2$), wrapping back to 1 after 61 is exceeded. Horizontally adjacent cells will differ by 2 (modulo 61 arithmetic). Vertically adjacent cells will differ by 14 (modulo 61 arithmetic). Consider a number of labellings with different I values as in Table 5.8. There is a unique rook-wise adjacency relationships in the first 6 cases. Clearly there are many more unique contexts if two orthogonal rook-wise adjacencies are used. By tiling a number of 7×7 grids with the first 6 increment values in the table we can expand the original area by a factor of 6 *i.e.* $\lfloor \sqrt{n} \rfloor - 1$ which is good enough to cover the laboratory. The difference between adjacent targets uniquely identifies the grid: one item of rook-wise adjacency information is required. Deriving location at the boundaries between the different 7×7 grids may be rather difficult. However, this basic technique given a rough idea of the scalability that is possible. There must be a better more general method.

Using Target Sequences

An arguably more elegant approach, and one less prone to error because an adjacent target need not always be detected, is to look at the sequence order of targets that form a straight line. An initial mathematical approach here is to look for sets of sequences (*i.e.* sets of permutations) of the numbers 0 to 63, that have no subsequence repetitions within 5 or less numbers, *e.g.*

```
        <------->
* * * * 1 * * 3 2 * * * * * * * * * * * * * * * *
```

and

```
                                    <------->
* * * * * * * * * * * * * * * * * * 1 * 3 * 2 * * * *
```

would not be allowed as different tilings. Suppose there are S sequences can we get of this form. If $S > 64$, and there is some way of distinguishing grid rows from grid columns, then a $S \times 64$ array of tiles could be usefully covered.

Target Dominoes

The following discussion is a little more mathematically rigorous, and tries to place some upper bounds on a particular adjacency method. Reduce the problem first to one dimension and look for sequences of elements with all unique doubletons. For sequences of $\{1, 2\}$, there is only one serious contender (not forgetting the constraint about not knowing the orientation): 1122. This is similar to a game of dominoes where double tiles are placed by overlapping numbers instead of just placing single tiles alongside each other. Each pair of adjacent squares forms a domino. The problem is reduced to making sure that each of the dominoes on the ceiling is unique. For sequences of $\{1, 2, 3\}$, laying dominoes as follows $11, 12, 22, 23, 33, 31$ gives the pattern 1122331 and exhausts all the dominoes numbered up to 3. For four different targets, it starts to get trickier and a possible layout is: 1122331442 but as the 43 tile has not been used, an opportunity has been missed. Another possibility of the same length involving this tile is: 1122334413. For dominoes numbered up to n, we have $d \equiv \frac{n(n+1)}{2}$ dominoes. Each domino covers 2 squares. Each square except for the end ones is covered 2 deep as in Figure 5.56. Let n_{single} be the number of squares covered by one domino, n_{double} be

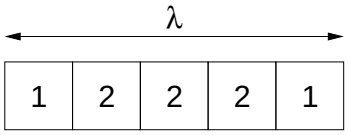

Figure 5.56: Tile thicknesses for 1D dominoes

the number of squares covered by two dominoes and λ be the maximal possible length of sequence. Then:

$$\begin{aligned} 1n_{single} + 2n_{double} &= 2d \\ \Rightarrow \quad 1 \times (\lambda - 2) + 2 \times 2 &= 2d \\ \Rightarrow \quad \lambda &= d + 1 \end{aligned}$$

So the maximal possible length of a sequence is simply $d+1$, but is it possible to use all the dominoes? It would appear to be quite easy for prime n. The simplest proof is a demonstration of a domino laying strategy. For $n = 3, d = 6, \lambda = 7$:

$$112233 \quad 1 \quad \text{(the space is just for ease of reading)}$$

For $n = 5, d = 15, \lambda = 16$:

$$1122334455 \quad 13524 \quad 1$$

For $n = 7, d = 28, \lambda = 29$:

$$11223344556677 \quad 1357246 \quad 1473625 \quad 1$$

The algorithm is transparent, for each block after the first, increment the increment until you start to go back on yourself. The evidence is convincing, for $n = 11, d = 66, \lambda = 67$:

$$1122334455667788 99aabb\,13579b2468a\,147a258b369\,15926a37b48\,16b5a493827\,1$$

This algorithm does not work for composite numbers because it uses the fact that in arithmetic modulo some prime p, for $r < p$, the sequence $r, 2r, 3r, ...pr$ is a permutation of $1, 2, ..., p$. There is probably a better algorithm which would work for all odd n but special cases would be needed when r divides n. For even numbers n, bigger than 4, it is impossible to place all the dominoes, since each number has to appear with each other on either side of it. For example, with $n = 4$, $d = 10$, $\lambda = 11$:

$$11223344 \quad 213$$ but the ends are not equal.

Whereas for $n = 6$, $d = 21$, $\lambda = 22$, try:

$$112233445566 \quad 135 \quad 246$$ but the technique starts to get in a mess.

So how does this generalise to two dimensional tiling? Well, each square (except the edges) must be covered by 4 dominoes as in Figure 5.57. Let n_{triple} be the number of squares covered by three dominoes, $n_{quadruple}$ be the number of squares covered by four dominoes and λ be the side length of the square area. Then:

$$\begin{aligned} 2n_{double} + 3n_{triple} + 4n_{quadruple} &= 2d \\ \Rightarrow \quad 2 \times 4 + 3 \times 4(\lambda - 2) + 4 \times (\lambda - 2)^2 &= 2d \\ \Rightarrow \quad \lambda^2 &= \frac{d + 1 + \sqrt{2d + 1}}{2} \\ \Rightarrow \quad \lambda^2 &\approx \frac{d}{2} \qquad \text{for 64 targets } d \text{ is } 2080 \end{aligned}$$

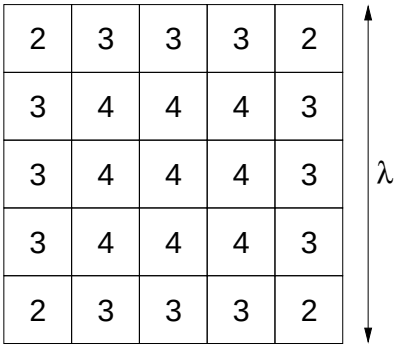

Figure 5.57: Tile thicknesses for 2D dominoes

So a simple upper bound on the area is $\frac{d}{2}$, but again the big question is whether this upper bound can be attained. No further thought was given to the problem except to note that for the circular targets with 64 different tiles, the area upper bound is around 1072.8. Compare this to the approach with six 7×7 meta-tiles giving an area of $6 \times 49 = 294$. This simplistic but semi-realisable scheme surprisingly achieves at least roughly 30% optimality.

61 is a prime and this would be a suitable number for a 1D tiling scheme as it involves minimal wastage (only 3 out of 64 targets are not used). Assuming the argument extends to 2D, with $n = 61$, $d = 1891$, and $area \approx 976.8$ (gives 351.6 square metres $\approx 18.8m \times 18.8m$) which is plenty big enough! Note, however, we are dealing with an upper bound and no attempt has been made to find suitable 2D tiling patterns. The earlier prime number result suggests using Mersenne primes which have the form $2^n - 1$, as n rings give 2^n IDs. The first few Mersenne primes are $3, 7, 31, 127, 8191$ and there are suitable ones in the right sort of range.

5.9.2 Related Forms of Target

A considerable amount of research has been carried out in the field of barcodes [61]. There are obvious similarities between barcodes and fiducials, the most significant being the fascinating challenge of encoding information in a 2D pattern. The high commercial interests are shown by the proprietary

nature of many systems. However, with great regret these barcodes were found not to be suitable for fiducials, because they are designed in general to be viewed both under ideal illumination conditions, and up-close with little distance variation. None of the barcodes appear to use RAGs for encoding. There is admittedly much spatial redundancy in a RAG target and barcodes are concerned instead with density of information storage.

The DataGlyph from Xerox PARC [62] is neither barcode nor fiducial but something in between. One of the most interesting and arguably zany recent ideas from Xerox PARC is that of "intelligent" paper. Development is currently very much under wraps, with few technical details being revealed. Using invisible marking in UV ink each sheet of intelligent paper is given a unique ID, and using a wand (with mini video camera at the tip) you can find out where you are on each sheet of intelligent paper (to within about 1/4"). The encoding density of the invisible information is 400 bytes per square inch. By selecting a "hot spot" on a book made from intelligent paper, information will be sent to the nearest Web-addressable screen. Possible applications include pronunciation guides for foreign language textbooks, more detailed information for a concise encyclopædia, *etc.* . Intelligent paper is a way of using a conventional book as an input device.

5.9.3 RAG Wallpaper

There are some conceptual ways to further extend the concept of RAG targets. Imagine a "RAG wallpaper" (*i.e.* a continuous sheet instead of discrete targets) which consists of hierarchical RAG patterns at various scales. At a large distance from the wallpaper (or equivalently with a poor quality camera) only gross detail can distinguished, and a "gross" position fix (or more strictly speaking one of accuracy commensurate to the positional information required or camera budget available) is the outcome. Close-up to the wallpaper very fine detail can be distinguished and a fine **and** unique position can be calculated. The requirement is for each zone of the wallpaper at each scale to yield a unique RAG. However, given the non-discrete nature of the wallpaper a segment of the continuum of the global RAG will be identified. To avoid graph searching, each section of the wallpaper could obey a RAG grammar. The RAG sentences would indicate location. One can hypothesise trees that form regular expressions in this grammar. The theoretical existence of a pattern that could fulfil such criteria would first have to be proven.

Clearly, the niche for the wallpaper is rather specialised (the ceilings or walls of studios and VR caves). Unlike Xerox PARC's intelligent paper, the wallpaper is scale independent and would overturn

the current dependency of requiring n targets to be visible which greatly constrains scale. The BBC virtual studio system [56] can just about manage with the theoretical minimum of 4 targets visible, but typical reliable operation is with around 12 targets. One of the requirements for the proposed RAG wallpaper is similar to one of the properties of Penrose Tiling Patterns [63], namely that though made up from a very small set of tiles (say 5) each region is unique and cannot be found elsewhere on the tessellation. What is required is a hierarchical fractal slant on this.

5.10 Conclusion

Surprisingly, there is very little information on the practicalities of finding targets and it was only at the end of writing the vision system herein described that a public domain system of comparable functionality emerged from the University of Washington [64]. Why is the image processing so under-represented? The suspicion is the difficulty in producing truly robust and reliable image processing software: this certainly reflects the experience of the author. While it may be possible to produce entirely satisfactory image processing under a certain set of conditions; change these conditions slightly and the whole system will usually fail. Moving through different target designs (circular barcodes, spirals and and concentric rings) and various image processing techniques (edge detection and the Hough transform) a robust solution based on region processing was produced where the target designs are derived from distinctive and distinguishable Region Adjacency Graphs.

Chapter 6

Geometric Vision Processing for Augmented Reality

6.1 Introduction

This chapter deals with the geometric calculations to establish camera position and orientation from an image of detected targets which have known real-world positions. These considerations (unlike target design) are independent of the image processing and so are presented separately in this chapter. The system that was developed is called the Video Positioning System (VPS). The chapter concludes with a quantitative evaluation of the accuracy of the VPS and describes its use in supporting AR applications.

6.2 Theory

An initial note on notation: a particular variable may be slightly modified and a primed symbol is used to represent the modified value. However, the un-primed symbol is used subsequently to represent the primed quantity, for simplicity of representation. To establish the position and orientation of a camera from the contents of the image, requires examining the mathematics of camera projection. In Figure 6.1 the centre of projection is at the origin, and the view direction of the camera is oriented along the z-axis. The image plane is at a distance f (the focal length) from the centre of projection. Note that for the sake of simplicity the image plane is in front of the centre of projection. In a real camera the image plane would be behind the centre of projection and the image would be inverted.

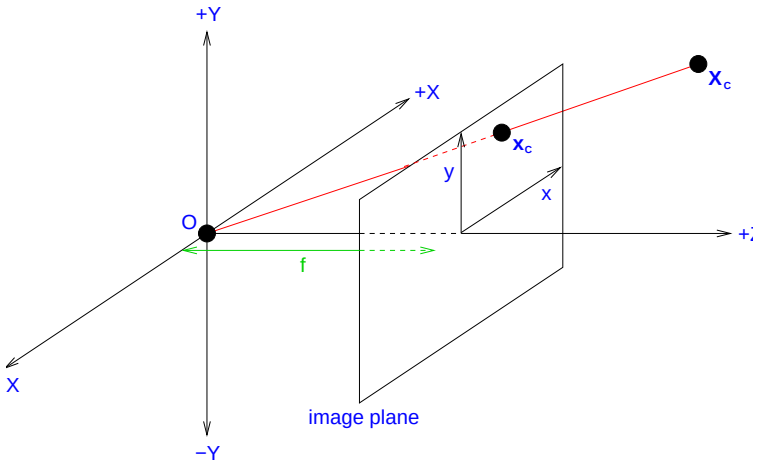

Figure 6.1: Camera projection

Using similar triangles, it can be seen that:

$$\frac{x_c}{X_c} = \frac{y_c}{Y_c} = \frac{z_c}{Z_c}$$

As $z_i = f$

$$x_c = f\frac{X_c}{Z_c} \quad \text{and} \quad y_c = f\frac{Y_c}{Z_c}$$

In matrix notation:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \frac{f}{Z_c} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$

$$\begin{bmatrix} x_c \\ y_c \\ f \end{bmatrix} = \frac{f}{Z_c} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$$

In general the camera cannot be so conveniently aligned to the coordinate system origin and before being projected in a camera-frame coordinate system, a point in an arbitrary world coordinate frame

must first be subject to a general Euclidean transform. Let the Euclidean transform between the camera and world coordinates be shown as:

$$\boldsymbol{X_c} = \boldsymbol{R}\boldsymbol{X_w} + \boldsymbol{T}$$

where $\boldsymbol{R} \in \mathbb{R}_{3\times3}$ is a rotation matrix and $\boldsymbol{T} \in \mathbb{R}_{3\times1}$ is a translation vector. Then

$$\boldsymbol{X_c} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{T} \\ \boldsymbol{0^T} & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_c \\ y_c \\ f \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{R} & \boldsymbol{T} \\ \boldsymbol{0^T} & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

However, the coordinates on the image plane are not yet the pixel coordinates within the image. In Figure 6.2, we assume that the pixel coordinates of the optical centre of the system are (u_0, v_0) and the pixel coordinates (u, v) of image point (x_c, y_c) are determined both by this offset and the pixels per unit length in the horizontal and vertical directions (k_u and k_v respectively which are characteristic of the CCD/imaging technology).

$$\begin{aligned} k_u x_c &= u - u_0 \\ k_v y_c &= v_0 - v \end{aligned}$$

Note that the pixel coordinates are obeying the standard convention of the v-axis oriented downwards. In matrix notation:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} fk_u & 0 & u_0 \\ 0 & -fk_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ f \end{bmatrix} = \boldsymbol{C} \begin{bmatrix} x_c \\ y_c \\ f \end{bmatrix}$$

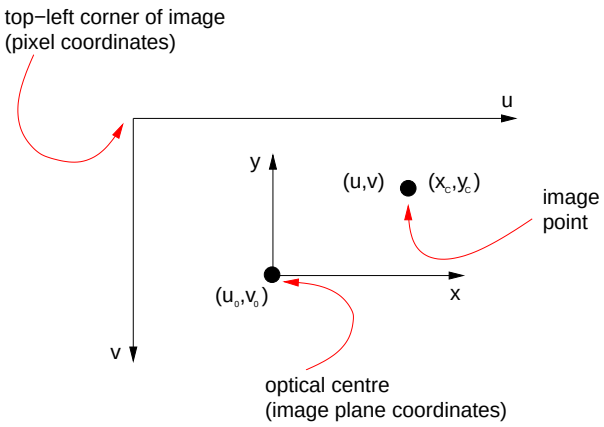

Figure 6.2: Conversion to pixel coordinates

$\boldsymbol{C} \in \mathbb{R}_{3\times 3}$ is an upper triangular matrix called the camera calibration matrix with the characteristic form:

$$\begin{bmatrix} a_u & 0 & u_0 \\ 0 & a_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

where $a_u = fk_u$, $a_v = -fk_v$. Once $\boldsymbol{C}$ is known the camera is called "calibrated". Obviously, for a zoom lens, rather than a single such matrix, there is a continuum of camera calibration matrices. Looking at the complete pipeline we have:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{C} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{R} & \boldsymbol{T} \\ \boldsymbol{0^T} & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

$$= \quad \boldsymbol{C}\,[\boldsymbol{R}|\boldsymbol{T}] \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

Overall therefore, the perspective projection from 3D space $\boldsymbol{X}$ to a 2D image can be compactly represented by the equation:

$$\boldsymbol{u} = \boldsymbol{P} \begin{bmatrix} \boldsymbol{X} \\ 1 \end{bmatrix} \quad \text{where} \quad \boldsymbol{P} = \boldsymbol{C}[\boldsymbol{R}|\boldsymbol{T}]$$

$\boldsymbol{P} \in \mathbb{R}_{3\times 4}$ is called the camera projection matrix. The reason homogeneous coordinates are used is to model the division in the perspective transform, as can be appreciated in the derivation. Further away objects appear smaller in inverse proportion to their distance from the camera.

$$\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}$$

$$\begin{aligned} u_i &= \frac{(p_{11}X_i + p_{12}Y_i + p_{13}Z_i + p_{14})}{(p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34})} \\ v_i &= \frac{(p_{21}X_i + p_{22}Y_i + p_{23}Z_i + p_{24})}{(p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34})} \end{aligned}$$

multiplying up:

$$\begin{aligned} u_i(p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34}) &= (p_{11}X_i + p_{12}Y_i + p_{13}Z_i + p_{14}) \\ v_i(p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34}) &= (p_{21}X_i + p_{22}Y_i + p_{23}Z_i + p_{24}) \end{aligned}$$

Rearranging into an appropriate matrix form for the unknowns p_{ij}:

$$\begin{bmatrix} & & & & & & \vdots & & & & & \\ -X_i & -Y_i & -Z_i & -1 & 0 & 0 & 0 & 0 & u_iX_i & u_iY_i & u_iZ_i & u_i \\ 0 & 0 & 0 & 0 & -X_i & -Y_i & -Z_i & -1 & v_iX_i & v_iY_i & v_iZ_i & v_i \\ & & & & & & \vdots & & & & & \end{bmatrix} \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \\ p_{31} \\ p_{32} \\ p_{33} \\ p_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

i.e. $\boldsymbol{Ap} = \mathbf{0}$ where $\boldsymbol{A} \in \mathbb{R}_{2n\times 12}$ is known, and $\boldsymbol{p} \in \mathbb{R}_{12\times 1}$ holds the 12 unknowns p_{ij}. n is the number of known point correspondences between the real world and the image. A point of such correspondence is called a *tie* point. $\boldsymbol{p}$ is not unique according to scale, so has 11 rather than 12 degrees of freedom. To be able to solve for $\boldsymbol{p}$ uniquely, in the general case, there would have to 11 equations and thus 6 point correspondences. Further point correspondences would (usefully) overspecify the problem. However, consider the case when all the points lie on a single plane. Here it is possible to shift the world coordinate system so, for example, $Z_i = 0$ for all points. We would solve for $\boldsymbol{p}$ in this coordinate system, and then transform this back to the original coordinate system. The mathematics

in this case become much simpler $\boldsymbol{A} \in \mathbb{R}_{2n\times 9}$:

$$\begin{bmatrix} & & & & \vdots & & & & \\ -X_i & -Y_c & -1 & 0 & 0 & 0 & x_cX_i & x_cY_c & x_c \\ 0 & 0 & 0 & -X_i & -Y_c & -1 & v_iX_i & v_iY_c & v_i \\ & & & & \vdots & & & & \end{bmatrix} \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \\ p_{31} \\ p_{32} \\ p_{33} \\ p_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The number of point correspondences required is now reduced to 4. Whether $\boldsymbol{A} \in \mathbb{R}_{2n\times 12}$ with $n \geq 6$ or $\boldsymbol{A} \in \mathbb{R}_{2n\times 9}$ with $n \geq 4$ the problem may be reduced to least squares to minimise the expression:

$$\sum |(u_i, v_i) - \boldsymbol{P}(X_i, Y_c, Z_i)|^2$$

One linear solution technique to is find the eigenvector corresponding to the least eigenvalue of $\boldsymbol{A}^T\boldsymbol{A}$. Another linear solution method is to find the singular value decomposition of $\boldsymbol{A}$. Any $m \times n$ matrix $\boldsymbol{A}$ where $m \geq n$, may be factorised

$$\boldsymbol{A} = \boldsymbol{U}(w_i)\boldsymbol{V}^T$$

where the columns of both $\boldsymbol{U} \in \mathbb{R}_{m\times m}$ and $\boldsymbol{V} \in \mathbb{R}_{n\times n}$ are orthonormal and $(w_i) \in \mathbb{R}_{n\times n}$ is a diagonal matrix where each $w_i \geq 0$. We find the lowest valued w_i, the solution is then the corresponding column of $\boldsymbol{V}$. Faugeras [65] suggests introducing known constraints on $\boldsymbol{P}$ in the solution. Let

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{q_1}^T & q_{14} \\ \boldsymbol{q_2}^T & q_{24} \\ \boldsymbol{q_3}^T & q_{34} \end{bmatrix}$$

Then the following two constraints can be proven [65]. The first is the simpler and is based on that fact that $\boldsymbol{q_3}$ can be shown to be a row of a rotation matrix and so has a norm of 1.

$$||\boldsymbol{q_3}|| = 1$$

$$(\boldsymbol{q_1} \times \boldsymbol{q_3}).(\boldsymbol{q_2} \times \boldsymbol{q_3}) = 0$$

Faugeras also suggests using a non-linear technique to find P that minimises:

$$\min_{\boldsymbol{P}} \sum_i |(u_i, v_i) - \boldsymbol{P}(X_i, Y_i, Z_i)|^2$$

using a linear solution as the starting point. However, Faugeras stresses that a simple linear approach is just as good as a non-linear approach providing effective sets of calibration targets are used. What constraints should be placed on the positions of these targets? Ideally the system of equations represented by $\boldsymbol{A}$ should have a unique solution, and this is the case if $rank(\boldsymbol{A})$ is $(m-1)$ where $\boldsymbol{A} \in \mathbb{R}_{2n \times m}$. When the reference points are in a plane, it can be shown that $rank(\boldsymbol{A})$ is 8, so moving to the lower dimension solution technique (m is 9 rather than 12) is actually necessary!

6.3 Implementation

The two linear solution techniques described above were implemented: using the eigenvector routine `jacobi` and the singular value decomposition routine `svdcmp` from the 'C' Numerical Recipes book [66]. The results from both approaches agreed to around six decimal places and acted as a useful cross-check. Using the calculated $\boldsymbol{P}$ to project the known coordinates of the targets into image space revealed that good visual registration was achieved in most circumstances — justifying the linear approach as sufficiently accurate for the application under consideration. Some caveats to this statement are presented in Section 6.5.8.

6.4 Factorising the Camera Projection Matrix

On initial reflection it may appear impossible to factorise the camera projection matrix $\boldsymbol{P}$ just established into the calibration matrix $\boldsymbol{C}$ and the camera orientation matrix $\boldsymbol{K} \equiv [\boldsymbol{R}|\boldsymbol{T}]$. For example, trying to factor the integer 24 into a unique multiplier and multiplicand is impossible! However, it is

fortunate that both $\boldsymbol{C}$ and $\boldsymbol{K}$ have sufficiently constrained forms, that it is possible to perform this factorisation [67]. Performing this factorisation is equivalent to solving simultaneously two classical problems of photogrammetry: namely the exterior orientation problem ($\boldsymbol{K}$ provides the position and orientation of the camera) and the interior orientation problem ($\boldsymbol{C}$ provides camera specific metrics such as focal length and field of view). Note the division into camera-specific and camera-independent measures. Both matrices have their uses, though in using the camera as a location device the only concern is solving the exterior orientation problem. For video-overlay AR applications, we would also wish to solve the interior orientation problem. How may this factorisation be performed?

6.4.1 Non-Coplanar Target Case

When $\boldsymbol{P}$ is derived from a non-coplanar set of targets, the elements of $\boldsymbol{C}$ and $\boldsymbol{K}$ may be computed from those of $\boldsymbol{P}$ as follows. Note that

$$P = \begin{bmatrix} \boldsymbol{q}_1 & q_{14} \\ \boldsymbol{q}_2 & q_{24} \\ \boldsymbol{q}_3 & q_{34} \end{bmatrix} \qquad C = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \qquad K = \begin{bmatrix} & \boldsymbol{r_1} & & t_x \\ & \boldsymbol{r_2} & & t_y \\ & \boldsymbol{r_3} & & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Initially $\boldsymbol{P}$ is scaled by the factor $-\frac{sgn(p_{34})}{(p_{31}^2+p_{32}^2+p_{33}^2)}$. The denominator "normalises" the matrix, and the sign term makes sure the orientation of the z-axis is correct *i.e.* t_z below is negative. Then

$$u_0 = \boldsymbol{q}_1^T \boldsymbol{q}_3 \qquad \alpha_u = \sqrt{(\boldsymbol{q}_1^T \boldsymbol{q}_1 - {u_0}^2)}$$

$$v_0 = \boldsymbol{q}_2^T \boldsymbol{q}_3 \qquad \alpha_v = \sqrt{(\boldsymbol{q}_2^T \boldsymbol{q}_2 - {v_0}^2)}$$

To make the results come out correctly, the matrix $\boldsymbol{K}$ obtained had to be premultiplied to form the matrix $\boldsymbol{K'}$ where:

$$\boldsymbol{K'} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{K}$$

Note that this "correction term" was introduced only after the true interpretation of $\boldsymbol{K}$ was established, through having to pass this matrix to the OpenGL [68] graphics library for rendering. The "after the fact" correction was retained, and it was not felt necessary to adjust earlier calculations. Faugeras himself is somewhat vague about the choice of signs (square roots come into the solution) and so it is unclear how to write deterministic code. Note that the equations presented above are slight variations of those in the book. However, with the signed pre-treatment of $\boldsymbol{P}$ and the signed post-treatment of $\boldsymbol{K}$, these equations have been shown to work, by experimentally investigating simple discriminating cases with the camera lens oriented in turn along each axis of the world coordinate system.

6.4.2 Coplanar Target Case

In the case of coplanar targets, there is insufficient information to perform the factorisation. In order to keep our positioning system running, we choose to use a pre-computed value of $\boldsymbol{C}$. Such a pre-computed value is obtained from a prior calibration run, see Section 6.7. Thus, for 2D targets the positioning system does not dynamically compute $\boldsymbol{C}$, and hence cannot be used with a zoom lens. As $\boldsymbol{P} = \boldsymbol{C}\boldsymbol{K}$, the solution technique for $\boldsymbol{K}$ is roughly equivalent to computing $\boldsymbol{P}\boldsymbol{C}^{-1}$ as both $\boldsymbol{C}$ and $\boldsymbol{P}$ are known. However, the mathematics are a little more involved and are detailed below. First the pre-computed values of a_u, a_v, u_0 and v_0, need to be scaled to be correct for the current resolution of video-grabbing. Fortunately, these scale completely linearly. Hence:

$$a'_u = \frac{w_g}{w_c} a_u \qquad u'_0 = \frac{w_g}{w_c} u_0$$

$$a'_v = \frac{h_g}{h_c} a_v \qquad v'_0 = \frac{h_g}{h_c} v_0$$

where w_g is the width of the images being grabbed, h_g is the height of the images being grabbed, w_g is the width of the calibration images, and h_g is the height of the calibration images. Two values of a

scale factor f are defined in an averaging calculation as below:

$$f = \pm \frac{\sum_{\substack{j=1 \\ j \neq planar\,dimension}}^{3} \frac{1}{{p_{3j}}^2 + (p_{2j} - v_0 p_{3j})^2 + (p_{2j} - u_0 p_{3j})^2}}{\sum_{\substack{j=1 \\ j \neq planar\,dimension}}^{3} 1}$$

Note that when the targets are on the $x = 0$, $y = 0$, $z = 0$ plane, the term corresponding to $j = 0$, $j = 1$, $j = 2$ respectively is omitted. $\boldsymbol{P} \in \mathbb{R}_{3\times4}$ is converted into $\boldsymbol{P} \in \mathbb{R}_{4\times4}$ thus:

$$p'_{ij} = \begin{cases} \delta_{ij} & \text{if } j = planar\,dimension \text{ or } i = 4 \\ f * p_{ij} & \text{otherwise} \end{cases}$$

For the reduced dimensionality case, the elements p_{ij} where j is the planar dimension are nonsense and have to be replaced, because their coefficients are zero in the system of linear equations.

$\boldsymbol{C} \in \mathbb{R}_{3x4}$ is converted into $\boldsymbol{C} \in \mathbb{R}_{4x4}$ thus:

$$\boldsymbol{C'} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note that matrices $\boldsymbol{P}$ and $\boldsymbol{C}$ are converted to a square 4×4 form, so inverses can be easily taken and multiplications are easily conformable. At this stage, there is a split and two possible methods were derived to compute $\boldsymbol{K}$.

Method 1

$$\begin{bmatrix} r_{1j} \\ r_{2j} \\ r_{3j} \end{bmatrix} = \begin{cases} \begin{bmatrix} r_{1((j+1)mod3)+1} \\ r_{2((j+1)mod3)+1} \\ r_{3((j+1)mod3)+1} \end{bmatrix} \times \begin{bmatrix} r_{1((j+1)mod3)+2} \\ r_{2((j+1)mod3)+2} \\ r_{3((j+1)mod3)+2} \end{bmatrix} & \text{if } j = planar\,dimension \\ \begin{bmatrix} p_{1j} - u_0 * p_3 j \\ p_{2j} - v_0 * p_3 j \\ p_{3j} \end{bmatrix} & \text{otherwise} \end{cases}$$

$$\begin{aligned} t_z &= p_{34} \\ t_y &= \frac{p_{24} - v_c * t_z}{a_v} \\ t_x &= \frac{p_{14} - u_c * t_z}{a_u} \end{aligned}$$

Method 2

$$\boldsymbol{K} = \boldsymbol{C}^{-1}\boldsymbol{P}$$

Then $\boldsymbol{K}$ is updated so:

$$\begin{bmatrix} k'_{1j} \\ k'_{2j} \\ k'_{3j} \end{bmatrix} = \begin{cases} \begin{bmatrix} k_{1((j+1)mod3)+1} \\ k_{2((j+1)mod3)+1} \\ k_{3((j+1)mod3)+1} \end{bmatrix} \times \begin{bmatrix} k_{1((j+1)mod3)+2} \\ k_{2((j+1)mod3)+2} \\ k_{3((j+1)mod3)+2} \end{bmatrix} & \text{if } j = planar\,dimension \\ \begin{bmatrix} k_{1j} \\ k_{2j} \\ k_{3j} \end{bmatrix} & \text{otherwise} \end{cases}$$

Method Choice

Both methods provide identical values for $\boldsymbol{K}$. In the end, the second method was chosen as it is the simpler: first an inversion and then the simple generation of an orthonormal column. Method 1

depends on the nature of the application, so the algorithm implementation is more likely to contain error. However, we do not yet have a unique value for $\boldsymbol{K}$, but 2 such values based on the two values of the factor f above.

Resolving the Mirror Ambiguity

Determining the correct sign of f is equivalent to resolving what the author calls the "mirror ambiguity". A sample detected target's position vector $\boldsymbol{p}$ is used and its normal vector $\boldsymbol{n}$ is computed. The normal for the target is computed by taking the cross product of the X-axis and the Y-axis on the plane of the sheet of paper on which the target is printed (remember both position and orientation of each target is known). The direction vector of the image under $\boldsymbol{K}$ of $\boldsymbol{n}$ positioned at $\boldsymbol{p}$ is computed as:

$$\boldsymbol{n'} = \boldsymbol{K}(\boldsymbol{p} + \boldsymbol{n}) - \boldsymbol{K}(\boldsymbol{p})$$

If the condition:

$$(n_x, n_y, n_z).(t_x, t_y, t_z) \leq 0$$

is true then the correct value for the sign of f has been chosen. The reason the calculation has been taken through with two values of different sign is that there is a genuine ambiguity in the system. With the targets on the same plane, a mirror image of the world (reflected in this plane) will also be a valid solution. To distinguish which of the two solutions is the world and which of the solutions is the mirror image, requires us to look at the fate of a target normal under the transformation. Basically, if the normal of the target is pointing towards the camera, then the solution is the correct one. This is because the targets are **not** printed identically on the back and because walls are not in general see-through, so if a target is facing away from the camera it will not be detected. In short, we use the knowledge that a particular target has been detected to resolve the ambiguity. The resolution of this mirror ambiguity took a considerable time, with many false starts! Finally, as in the 3D case $\boldsymbol{K}$ established by either of these two methods is multiplied by a sign-correction matrix.

$$\boldsymbol{K'} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \text{-}1 & 0 & 0 \\ 0 & 0 & \text{-}1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{K}$$

6.5 Factors Affecting Performance of the VPS

The factors that influence the accuracy of the VPS are enumerated and then discussed below.

1. number of targets in view
2. distance of targets
3. accuracy to which target positions are known
4. accuracy of region centroid location
5. layout of visible targets
6. resolution of imaging technology
7. quality of lens
8. 2D/3D operation
9. nature of augmentation
10. sophistication of camera model

6.5.1 Number of Targets in View

The camera available for the development of the VPS was rather less wideangle than was wished. As a result, to ensure enough targets were continually in view to a camera strapped to the side of a user's head and pointing upwards, every ceiling tile in the laboratory had a target attached. This represented a considerable investment in effort, as there are roughly 250 ceiling tiles in the laboratory. It took two days to attach the targets to the ceiling, and almost as long to paint the heads of 1000 multi-coloured drawing pins matt black! Permission was not given to stick the targets to the ceiling, so they had to be pinned into position with the drawing-pins hidden in the black parts of the target patterns. More targets in view will improve the accuracy, but what is the critical number of targets to achieve the accuracy required for our application? A quantitative study was made, see Section 6.8. A particular advantage of the VPS is that it can operate with just a single target, because each target provides 4 tie-points. Under typical conditions, visual registration will suffer in this circumstance, but with two or more targets in view, registration is visually "locked",

6.5.2 Distance of Targets

Further than a certain distance away from the camera, targets will no longer be located by the VPS. This distance will decrease as the obliqueness of the target to the camera axis increases. This distance will also decrease (up to a certain limit) as the focal length of the lens decreases. Clearly, the smaller the area of the image taken up by a target the less the four tie points of this target can contribute to the accuracy of the overall solution.

6.5.3 Accuracy to which Target Positions are Known

The repeat of the ceiling tiles was found to be exactly 60cm, and it was assumed therefore that the targets were placed on an ideal 60cm rectilinear grid. In reality, the ceiling will contain distortions. There are techniques to refine determination of approximately known target positions [56] [31]. A video (or sequence of images) is taken that contains every ceiling tile and "covers" the ceiling in the following sense. Let ρ represent the binary relation "in the same image as" and let a and b be arbitrary targets, then there exists an $n \geq 0$ such that

$$a(\rho)^n b$$

A global minimisation is then performed across these images to determine the actual positions of the targets. However, the results obtained from the VPS were sufficiently good to not require this additional set-up stage. This technique is probably the principal one "waiting in the wings" to refine accuracy should it prove necessary.

6.5.4 Layout of Visible Targets

The geometric configuration of targets affects accuracy in a complex manner much as the configuration of satellites in a GPS position fix, see Chapter 3. Two particular configurations are quantitatively studied in 6.8.

6.5.5 Accuracy of Region Centroid Location

The RAG-based VPS locates the centre of regions, by calculating the centroid of the region as it appears on the image plane. Figure 6.3 shows a perspective distorted rectangular region, and shows that the centroid of the region in 3D space (which is the tie point) does not correspond to the centroid

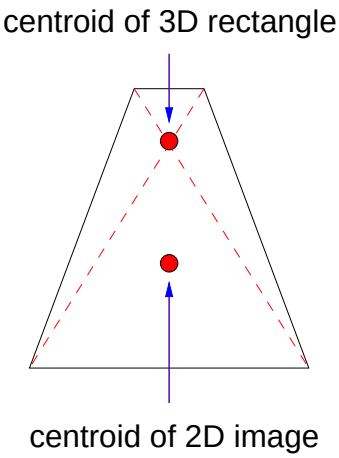

Figure 6.3: Region centroid not preserved by perspective projection

of the region in image space. Some inaccuracy is introduced by using this incorrectly calculated centre for the rectangular region. However, [69] establishes that despite this, it is more accurate to use such a centroid than by using the crossing point of detected corners. Note also, that the adaptive filtering employed may cause the detected region to be larger or smaller than its actual size. However, provided this occurs symmetrically around the boundary of the region, then the centroid will be unaffected as a measure of the region's centre.

The author believes the accuracy of centroid location can be significantly sub-pixel. For the circular targets, two measurements were taken from the calculated centroid of the target, in diametrically opposed directions, to the calculated edge of the target. The two radii agreed to within a hundredth of a pixel.

From an oblique angle the edges of the circular target rings were always found to be shifted slightly to one side. In fact this was the biggest source of error in identifying the targets correctly. Eventually from the direction of the shift, this was attributed to the "target sag" of drawing pin technology (*cf.* paste). This is a further argument for the "non-geometric" RAG targets, where shape distortion does not impact on identification.

A second pass may be possible. Once $\boldsymbol{P}$ is established, the degree of perspective distortion for each target can be estimated and corrected tie points established. These corrected positions can then

be used to produce a refinement $\boldsymbol{P}'$. Again it can be reported that the registration achieved was sufficiently good, that this step was not taken.

6.5.6 Resolution of Imaging Technology

A PAL-based system may sample at a maximum effective resolution of 576×768. At this resolution using a 266 MHz Pentium, it takes around one second to process a frame. The system was used typically at half-PAL, so four position fixes per second were calculated. A second advantage of working at half-PAL is that there is automatically no "tearing" between the odd-and-even scan lines of the image. That a single standard PAL image consists of two fields (both odd and even) taken at two different times, is apparent when the camera is moving rapidly. This at times interferes with the operation of the VPS. The blurring caused by fast camera motion can be much reduced by switching to a fast shutter speed. Modern CCDs are very sensitive, and can work with exposure times of around a millisecond. The shearing though still remains. However, by switching the camera to "progressive scan" mode the shearing can be eliminated. In progressive scan both fields are take from a single exposure, though not all PAL cameras can operate in this way.

The goals of having as many targets in the scene as possible and of each target being as large as possible in the image are antagonistic. The ideal situation would be to capture an image covering 2π steradians, or even approaching 4π steradians, at extremely high resolution. One could hypothesise such a specially designed input device for the VPS. However, capturing at half-PAL with the default lens of the development camera does not appear to be a limiting factor in obtaining positional accuracy with the VPS.

6.5.7 Quality of Lens

A high quality lens is regarded as that which most closely approximates a pin-hole camera: where straight lines in the real world will map to straight lines in the image. Good quality lenses are largely free of distortion. However, with very wide angle lenses, it is especially expensive to avoid the fish-eye effect, and many cheap webcams exhibit this form of axial distortion. Figure 6.4 shows the image from a small cheap video camera. It is noticeable that the grid lines formed by the ceiling tiles are curved. The camera used throughout the development and testing of the VPS was, in contrast, of sufficiently good quality that it was not felt necessary to compensate for lens distortion. As an experiment the VPS was connected to this cheap video camera (though the calibration used was obtained from the

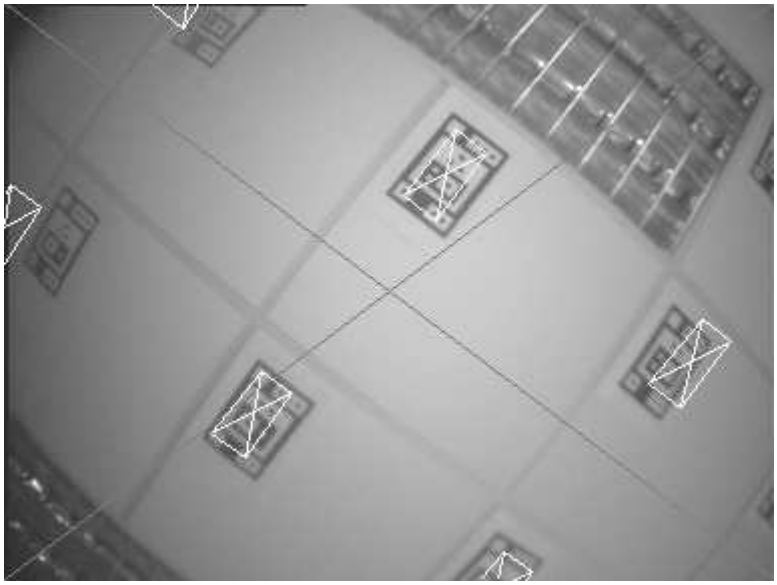

Figure 6.4: Distortion of a cheap wideangle lens

more expensive development camera). It can be seen that the lack of registration of the virtual and real targets is largely due to the effect of lens distortion. In fact, even with incorrect calibration and a distorting lens, the VPS still provides reasonable position fixes. There is a surprising degree of robustness though no qualitative investigation was made into the nature of the degradation.

The VPS does check the closeness of the fit of the actual target position in the image and the corresponding computed targets positions using $\boldsymbol{P}$. If the average squared distance is greater than a specified limit, then a warning flag is raised that the position reported is "noisy". This generally indicates some gross error in the system so the position should not be used. However, a warning also arose when the inappropriate calibration matrix is used for a lens, or that lens was in itself distorting. Generally, the limit for the root of the sum of squares of offset errors was supplied as 5. With the VPS is correct operation the agreement between the actual and computed position of the targets was much less than 5 pixels.

6.5.8 2D/3D Operation

The geometric processing goes down two distinct paths according to whether the targets are coplanar or not. Traditional wisdom says that targets in a 3D configuration provide greater accuracy, but results from the VPS undermine rather than directly refute this claim. In 3D processing an image-by-image

camera calibration is computed ($\boldsymbol{C}_{3D}$ and $\boldsymbol{K}_{3D}$ are established from $\boldsymbol{P}$); in 2D processing a previously determined camera calibration matrix is used ($\boldsymbol{C}_{2D}$ is known, $\boldsymbol{K}_{2D}$ is established from $\boldsymbol{P}$ and $\boldsymbol{C}_{2D}$). As an experiment, in the 3D case, both 3D and 2D processing were applied. Whereas the products $\boldsymbol{C}_{2D}\boldsymbol{K}_{2D}$ and $\boldsymbol{C}_{3D}\boldsymbol{K}_{3D}$ will accurately reproduce the effect of $\boldsymbol{P}$, the product $\boldsymbol{C}_{2D}\boldsymbol{K}_{3D}$ will give a jittery registration that jumps around to a small extent. As we know that $\boldsymbol{C}_{2D}$ is correct (it is time averaged), this implies that $\boldsymbol{K}_{3D}$ is wrong *i.e.* an individual factorisation is not necessarily that accurate.

In this experiment only a couple of targets were in use, and it may well be that perfectly accurate factorisations occur with large numbers of targets. However, we are investigating situations that will most often occur in practical use of the VPS. The algorithms and code were checked to try to find a mistake that would account for the imperfect factorisation. Finally, it was concluded that these inaccuracies were inherent in the Faugeras method itself, probably magnifying small errors in $\boldsymbol{P}$. It is an open question whether a non-linear determination of $\boldsymbol{P}$ would improve the factorisation. However, as a linearly determined $\boldsymbol{P}$ provides perfectly accurate visual registration, it is suspected to be unlikely.

The recommendation is thus always to use a precomputed camera calibration matrix where possible, and where this is not possible to use a running average of $\boldsymbol{C}_{3D}$ for $\boldsymbol{C}_{2D}$. The 3D method (Faugeras) is recommended for calibration only, where it is necessary, and calibration should be performed over multiple frames to obtain a good estimate of $\boldsymbol{C}$. The VPS can work (through command line switches) in calibration mode or from a previously computed calibration stored in a file.

The experiments in Section 6.8 confirm superior accuracy for 2D target sets over the 3D target sets, though like is not being compared for like, because two different processing paths are used. Experiments conducted by the BBC [56] have demonstrated that in fair trial, that 3D target sets do provide somewhat better accuracy than 2D targets sets. An intuitive explanation is that a pan and a translation are difficult to distinguish using a 2D target set because there is no relative movement (to first order) between the targets. This trial was not conducted for VPS as the experimental design cannot be perfect *i.e.* there is no such thing as an equivalent 2D and 3D target set to make a direct comparison. However, suppose we arrange alternate ceiling targets (*i.e.* those in a checkerboard pattern amongst the grid of ceiling tiles) to be lowered by 50cm. Allow a comparison to be made. Then reduce this distance to 20cm and carry out a second comparison, *etc.* . This is a subtly different but valid experiment that explores limiting behaviour, depending on the height variation amongst notionally "coplanar" targets. This is a good practical way of determining whether 3D targets are useful - they are! There are clear

mechanical problems in establishing the described 3D set-up, and this is another reason why the VPS was not tested in this way. While the BBC temporarily dismantled their virtual studio, they tested out their system in an office environment, with their targets attached to ceiling tiles (*i.e.* a 2D target set). The resulting positioning was satisfactory, though admittedly many more targets were in view than normal. The findings from the VPS shows similarly that under most circumstances a 2D target set is adequate for AR. The VPS will automatically switch between 3D and 2D operation depending on the nature of the targets in view, though it can also be forced to always use 2D-style processing (recommended).

6.5.9 Nature of Augmentation

Augmenting the image used to determine position is a simpler proposition than augmenting a secondary (say user's) view that is off the camera axis and perhaps displaced somewhat in space. In the former case only $\boldsymbol{P}$ needs to be determined, so the system is not strictly operating as a positional device. In the latter case $\boldsymbol{K}$ must be determined. Particularly when 3D processing is used, the additional factorisation step may introduce further inaccuracy as described above.

6.5.10 Sophistication of Camera Model

The camera model we employed only used 4 parameters u_0, v_0, α_u, α_v. It is possible to also model and then correct for lens distortion. A typical example is Tsai's camera model [70]. Public domain code to calibrate a camera against this model exists on the Internet. Tsai's camera has one more parameter κ_1 which is the first order radial lens distortion coefficient. Corrections have to be added to the uncorrected image coordinates (u, v) in order to obtain the "true" image plain coordinates thus:

$$\begin{aligned} u' &= u + \delta u \\ v' &= v + \delta v \end{aligned}$$

The corrections are modelled in terms of an even polynomial of the distance to the optical centre of the camera (distortions are generally radially symmetric).

$$\begin{aligned} \delta u &= (u - u_0)(\kappa_1 r^2 + \kappa_2 r^4 + \kappa_3 r^6) \\ \delta v &= (v - v_0)(\kappa_1 r^2 + \kappa_2 r^4 + \kappa_3 r^6) \end{aligned}$$

where $r^2 = (u - u_0)^2 + (v - v_0)^2$. Tsai only models the first second degree term; few models extend beyond the third sixth degree term shown.

A second brute force approach to correct for lens distortion is to perform an initial warping of the captured bitmap image that undoes the effect of lens distortion. For example, if the distorted image of a rectilinear grid of lines were subjected to this warping, then the rectilinear nature of the grid would be restored. Indeed, this is probably the manner in which the corrective warp would be established. Any distortion can be captured and corrected for by this method - not just that modelled by the single simple distortion parameter of Tsai. However, there is the computational overhead of a potentially complex warp per frame, and interpolation artifacts will be introduced. A much more lightweight and therefore satisfactory adaptation of this technique is to merely warp the coordinates of the extracted points.

Both the Tsai and brute force methods assume that the amount of camera distortion is independent of the distance from the camera to the object. Clearly, for distorting cameras some correction must be made to give accurate results. It is possible, though unlikely, that much additional accuracy will be gained in the case of better quality lenses where the distortion is below that perceptible by the human eye *i.e.* good visual registration is achieved across the frame. As a brief experiment, the camera model used was simplified to just two parameters α_u and α_v. It was assumed that (u_0, v_0) was the centre of the image. Registration was then systematically out by, typically, one sixth of the dimension of the image. The conclusion is that any successful camera model must incorporate parameters for the optical centre.

6.6 Refinement Algorithm

When circular targets were in use by the VPS, it was possible that a target, although recognised, would be misidentified. Misidentifications were generally caught, as the different algorithms to establish target ID would give different answers. Is it somehow possible to still use the information from a misidentified target in a position fix? The following two pass algorithm was developed.

1. Use the correctly identified targets to establish $\boldsymbol{P}$
2. For each misidentified target position in the image establish which of the remaining target IDs maps closest under $\boldsymbol{P}$.

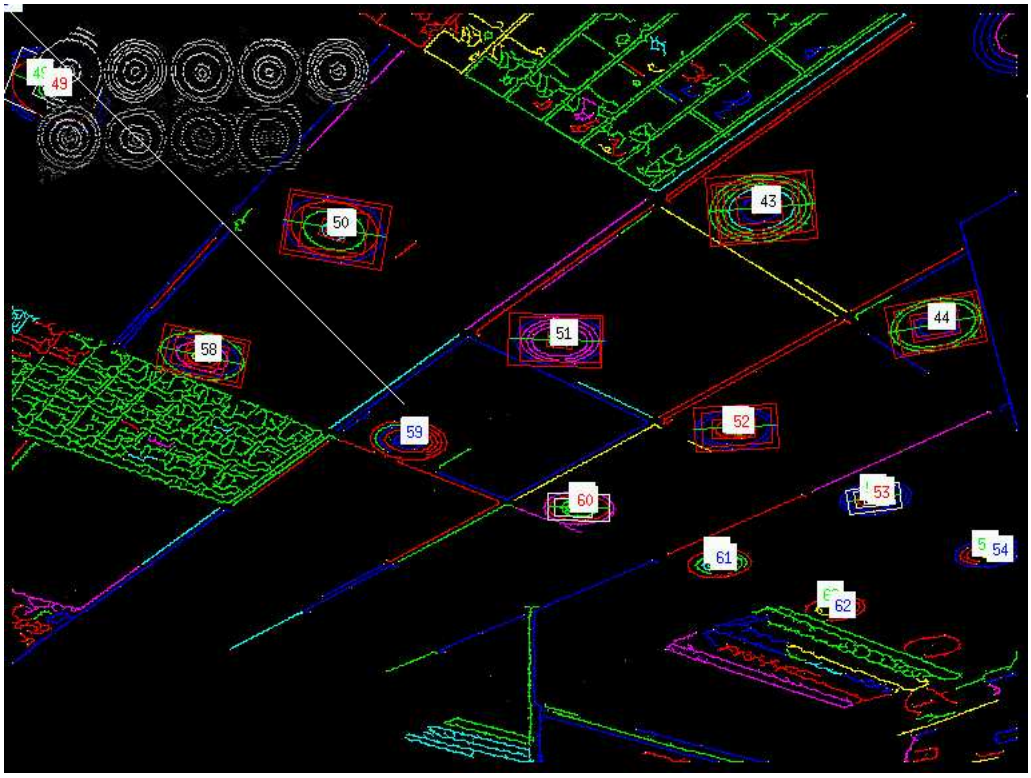

Figure 6.5: Scene with circular targets

3. Collect all misidentified targets for which an ID has been successfully guessed. The test for success is if the distance between the expected and actual image positions is below a certain (user-specified) distance.

4. Use the originally correctly identified set of targets, and the newly correctly identified set of targets (once misidentified) to establish $\boldsymbol{P}'$.

$\boldsymbol{P}'$ is a refinement of $\boldsymbol{P}$, and this approach worked well in practice. In Figure 6.5 targets 43, 44, 50, 51 and 58 (coloured by the VPS in black) have been correctly identified in the first pass. It is noticeable these are closest to the camera. Targets 49, 52, 53 and 60 (coloured by the VPS in red) have been correctly identified but only in the second pass. Note that there are no remaining incorrectly identified targets. Aside from these black and red labels, which have been placed on top, each target has been labelled where it would be expected to be found according to P (labelled is green) and where it would be expected to be found according to P' (labelled in blue). Targets 54, 61, 62 have not been found though they are known to be within the image (coloured by the VPS in green and then blue).

The displacement between labels for the same target, gives an indication of the magnitude of the refinement $\boldsymbol{P} \rightarrow \boldsymbol{P}'$. As can be seen from Figure 6.5, this represents a useful increase in accuracy.

Although the RAG targets do not suffer from the problem of misidentification, the hooks exist within the VPS system to support this type of refinement. If a target is given an ID of -1, then it represents a located misidentified target and subsequent stages of VPS processing will pull in this refinement processing. The latest RAG-based version of the VPS does not produce target IDs of -1, but the modular design does not inhibit future systems doing so. There are some advantages to the gradual degradation with distance of the circular target approach.

6.7 Calibration

Initial calibrations were accomplished by sticking targets on both walls and ceilings to produce a 3D target set. To overcome the inaccuracies in this approach, where targets were simply positioned manually using a metal tape rule, a dedicated calibration rig was constructed consisting of two printed A4 sheets holding 6 targets each, jointed accurately at 90°, by a specially constructed plastic frame. The calibration rig is shown in Figure 6.6. A calibration matrix was established by time averaging the per-frame calibration matrices.

6.8 Accuracy

This section qualitatively assesses the positional and angular accuracy of the VPS. No single figure can be provided as the accuracy depends on many factors as described above. Instead, particular set-ups were studied. Two targets sets with different characteristics were chosen, representing two different environments for VPS operation. One set of targets used was that on the 3D calibration rig (target positions known accurately, small in physical extent, non-coplanar); the other set of targets chosen were the ones stuck to the ceiling tiles in the laboratory (target positions not known accurately, large in physical extent, coplanar). The former scenario allows inherent errors in the VPS to be studied under controlled conditions; the latter scenario represents a practical set-up and indicates results which might be achievable “in the field”.

Determining angular accuracy is made more difficult by the variety of complex and unintuitive conventions for representing 3D orientation: a rotation matrix; a direction vector plus angle of rotation around this axis; Euler angles; and Quaternions. The Euler representation consists of three numbers,

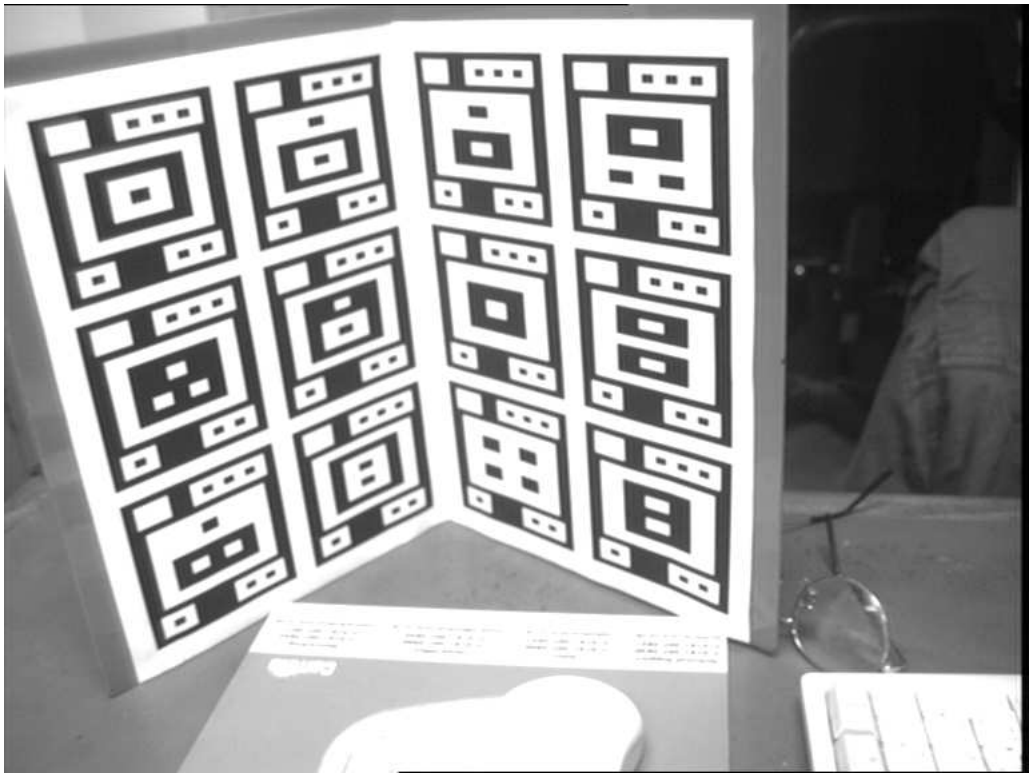

Figure 6.6: Calibration rig

based on the fact that an arbitrary 3D rotation can be described by three successive traditional 2D rotations about the coordinate axes. The individual numbers correspond to conventional angles which can be expressed in degrees, and so this format was chosen to make error values comprehensible. There are a variety of conventions for Euler angles, depending on the axes about which the rotations are carried out. We have chosen the so-called "x-convention", used in the analysis of gyroscopic motion, where the Euler angles (ϕ, θ, ψ) represent a first rotation of ϕ about the z-axis; a second rotation of θ about the y-axis; and a third rotation of ψ about the z-axis (again). A Quaternion is the 4-dimensional analogue of a 2-dimensional complex number, and can usefully represent 3D rotations in other circumstances. Software was written to interconvert between all four angular representations described: the Euler angles presented are derived from the VPS rotation matrix.

standard deviations in static VPS measurements over 99 frames					
X (m)	Y (m)	Z (m)	ϕ	θ	ψ
0.00026	0.000046	0.00059	0.021°	0.025°	0.017°

Table 6.1: VPS static accuracy: calibration targets

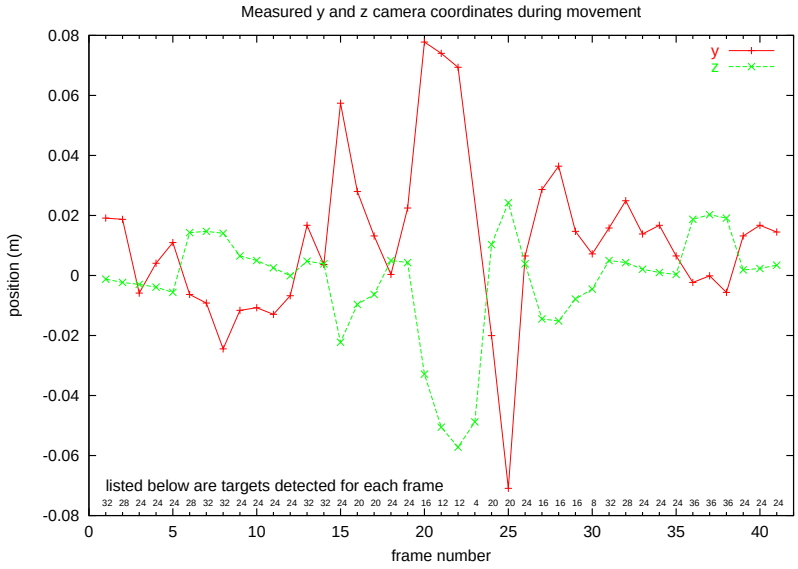

Figure 6.7: VPS perpendicular translational results: ceiling targets

6.8.1 Static Accuracy: Calibration Targets

The random noise in the position and angular measurements returned from the VPS was studied. With the 12 targets on the calibration rig in view and the rig occupying around half the image, the standard deviations derived from 99 consecutive frames are shown in Table 6.1. The low values indicate the stability of the system.

6.8.2 Translational Dynamic Accuracy: Ceiling Targets

The camera was pointed upwards to view the targets on the ceiling, and then moved along a straight line parallel to the x-axis over a distance of 4 m in 10 cm increments. Marks were measured out on the floor to guide this movement. The measured y and z positions are plotted in Figure 6.7 with the average set to zero for clarity. The difference between successive x positions is shown in Figure 6.8. The noise in the dynamic case has increased quite significantly and does not appear to be random.

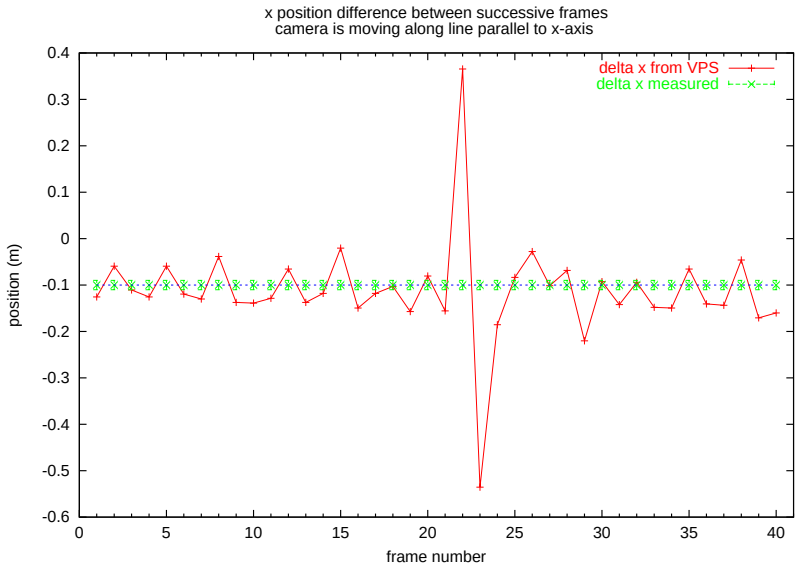

Figure 6.8: VPS parallel translational results: ceiling targets

standard deviations in dynamic VPS test (measurements over 41 frames)			
data set	$\sigma_{\Delta x}$ (m)	σ_y (m)	σ_z (m)
all	0.11	0.086	0.017
outlier removed	0.044	0.027	0.016

Table 6.2: VPS translational accuracy: ceiling targets

To check this a best-fit line straight was derived for each of the x, y and z coordinates. The error residuals between this best fit and the measured values were calculated. x and y residuals correlated to a high degree with a Pearson's coefficient of -0.94. Both from this correlation coefficient and a visual inspection of the graphs the errors appear more systematic than random. It is notable that the outliers occur when only one target is identified so only 4 points are available to the VPS. The standard deviations before and after the removal of the most significant outlier are given in Table 6.2. The beneficial effect on accuracy of greater target number is investigated in greater detail below, but is a low target count the only factor affecting performance of the VPS? Even after removal of points computed with only a low number of targets, there was still significant error in the system. Perhaps this is due to the fact that the absolute positions of the targets in the real world have only been estimated rather than measured?

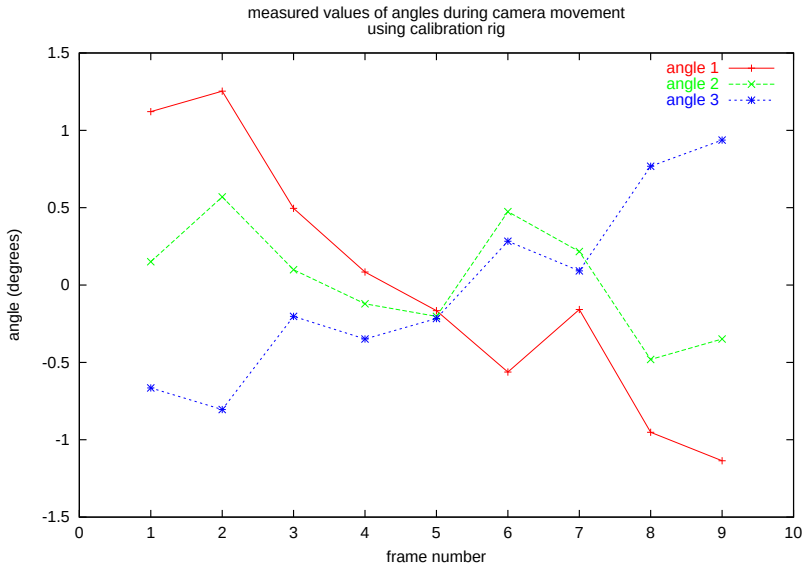

Figure 6.9: Measured angles during camera translation: calibration targets

standard deviations in dynamic VPS test (measurements over 9 frames)				
horizontal distance (m)	height (m)	ϕ	θ	ψ
0.015	0.0017	0.81°	0.37°	0.58°

Table 6.3: VPS translational accuracy: calibration targets

6.8.3 Translational Dynamic Accuracy: Calibration Targets

The camera was pointed at the calibration rig and them moved away in a straight line across the floor for a distance of 80 cm. Measurements were taken every 10 cm. The measured angles and heights are shown in Figure 6.9 and 6.10 with the mean adjusted to be zero. Figure 6.11 shows the horizontal distance moved between frames, calculated from the difference in x and y coordinates. The performance is very accurate in comparison to the ceiling target case. This is attributable to a better knowledge of the absolute targets positions and the higher number of targets used.

6.8.4 Rotational Dynamic Accuracy: Calibration Targets

The camera was fixed to a rotating turntable and the turntable was rotated $55°$ clockwise with measurements taken every $5°$. The camera was then rotated a second time in the opposite direction (anti-

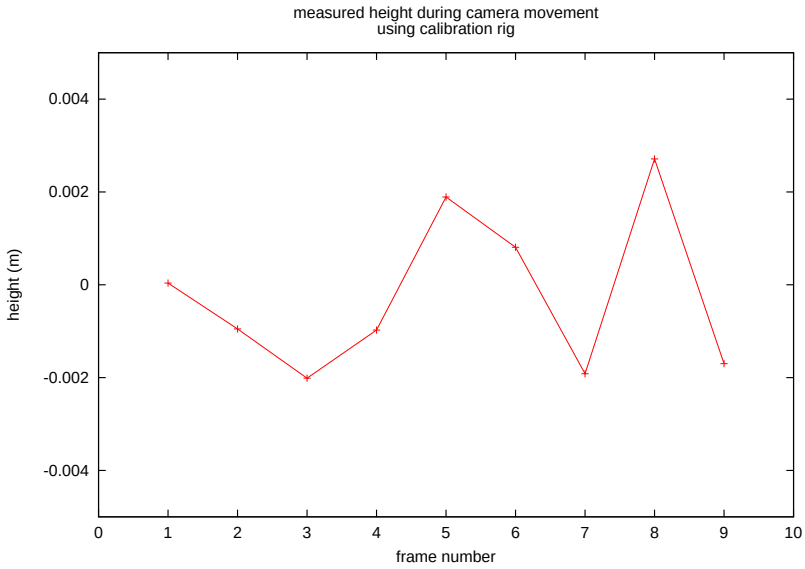

Figure 6.10: Measured height during camera translation: calibration targets

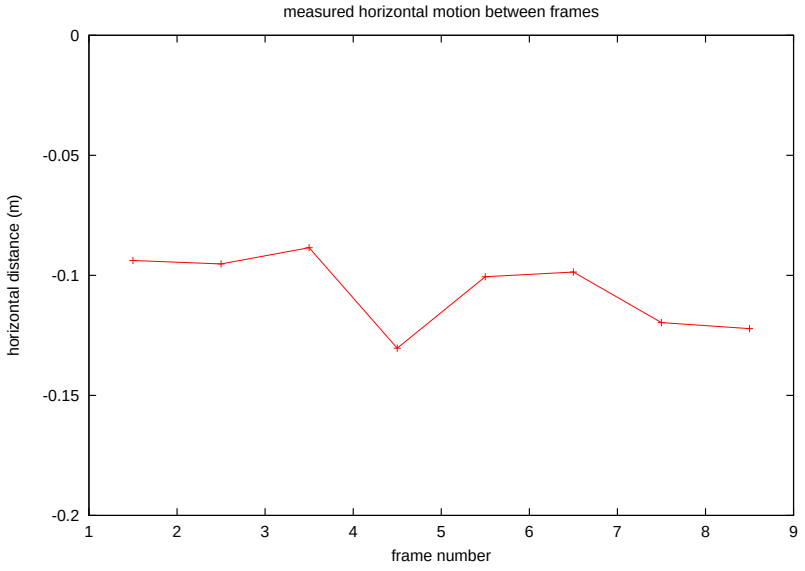

Figure 6.11: Measured horizontal distance during camera translation: calibration targets

frame	no of points	coplanar	T_x (m)	T_y (m)	T_z (m)	ψ	θ	ϕ
1	8	1	0.971	0.278	0.722	102.8°	31.4°	97.8°
2	20	0	0.941	0.273	0.770	93.7°	35.6°	100.0°
3	28	0	0.871	0.271	0.686	90.9°	39.0°	97.1°
4	36	0	0.902	0.269	0.694	89.5°	44.7°	93.3°
5	40	0	0.935	0.272	0.719	89.1°	47.2°	88.8°
6	40	0	0.934	0.271	0.709	88.5°	50.9°	84.1°
7	44	0	0.932	0.268	0.679	89.2°	56.6°	78.4°
8	40	0	0.930	0.269	0.666	89.4°	60.1°	73.4°
9	36	0	0.913	0.268	0.649	90.0°	63.5°	68.0°
10	24	0	0.897	0.273	0.632	91.7°	65.1°	62.4°
11	8	1	0.866	0.250	0.842	94.4°	69.4°	55.8°
12	4	1	0.766	0.531	0.807	73.6°	74.1°	68.5°

Table 6.4: Clockwise camera rotation: calibration targets

clockwise) with measurements repeated every 5°. Great care was taken to ensure that the principal axis of the camera was at a normal to the turntable and aligned with the centre of the turntable. The angular measurements in Figure 6.12 and positional measurements in Figure 6.13 show the clockwise and anti-clockwise passes produce similar results suggesting the experimental set-up was very accurate. The source data in Tables 6.4 and 6.5 indicate that ψ stays roughly constant, whereas θ increases by roughly 5° per frame and ϕ decreases by roughly 5° per frame. However, the graphs are far from the straight lines that might have been expected, though this is more nearly true while the system is in non-coplanar operation. The expectation also might have been that only one angle was changing while the other two stayed constant. The discrepancy is probably due to the lack of alignment of the camera on the turntable: it is difficult to know the exact physical whereabouts of the optical centre in relation to the body of a camera. Furthermore, Euler angles are not the most intuitive way of understanding orientation in any particular situation and it is difficult to know what values to expect. Though the results are not particularly conclusive from this experiment, they indicate good reproducibility of results.

6.8.5 Varying the Number of Targets: Calibration Targets

The number of targets was varied through software simulation, as this is easier than trying to gather representative data with different numbers of targets. By using this approach the problem was also solved of trying to gather images of different numbers of targets that were somehow "equivalent" in all

frame	no of points	coplanar	T_x (m)	T_y (m)	T_z (m)	ψ	θ	ϕ
1	4	1	0.759	0.460	0.843	78.4°	70.9°	65.7°
2	12	1	0.916	0.318	0.830	80.0 °	71.5°	58.0°
3	24	0	0.904	0.274	0.637	91.5 °	65.6°	62.5°
4	36	0	0.931	0.270	0.661	89.9 °	63.6°	68.1°
5	40	0	0.930	0.269	0.667	89.4 °	59.9°	73.4°
6	44	0	0.938	0.269	0.681	89.4 °	56.8°	78.4°
7	40	0	0.943	0.271	0.716	88.7 °	50.9°	83.9°
8	40	0	0.935	0.271	0.719	88.9 °	47.5°	88.7°
9	36	0	0.926	0.272	0.716	89.4 °	44.7°	93.5°
10	28	0	0.886	0.272	0.698	91.7 °	39.1°	96.7°
11	20	0	0.951	0.273	0.774	94.0 °	36.2°	100.0°
12	8	1	0.973	0.301	0.730	100.0 °	31.1°	100.5°

Table 6.5: Anti-clockwise camera rotation: calibration targets

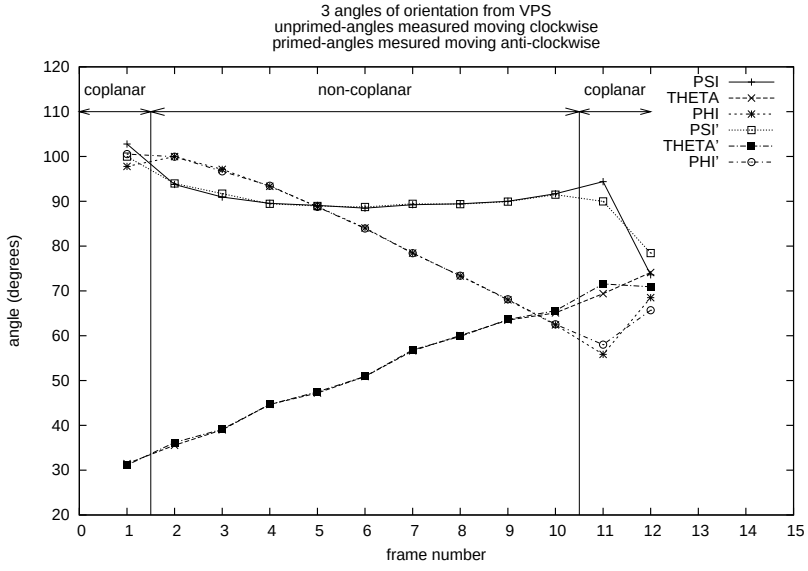

Figure 6.12: Measured angles during camera rotation: calibration targets

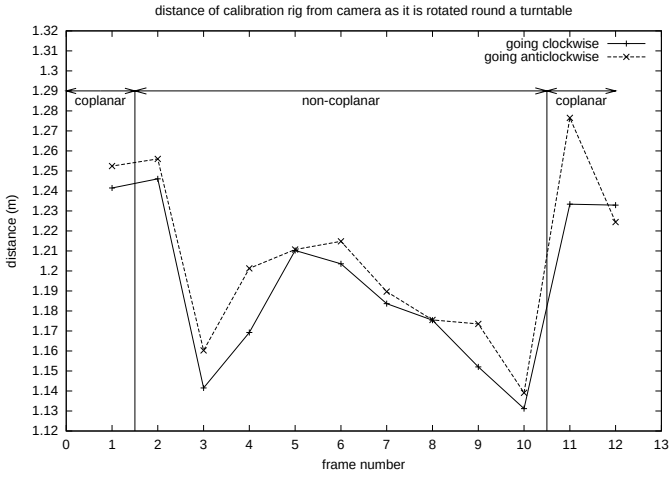

Figure 6.13: Measured horizontal distance during camera rotation: calibration targets

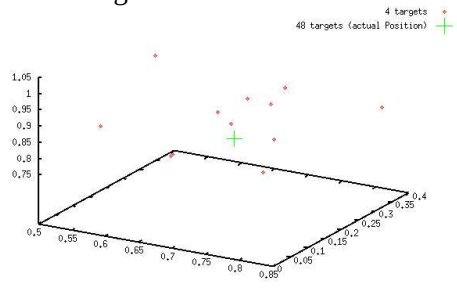

Figure 6.14: 4 targets

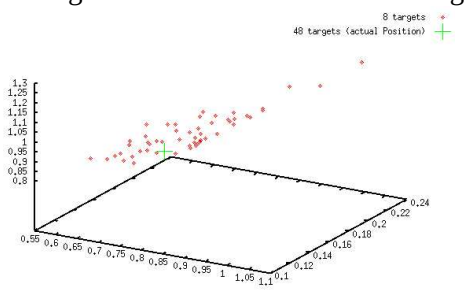

Figure 6.15: 8 targets

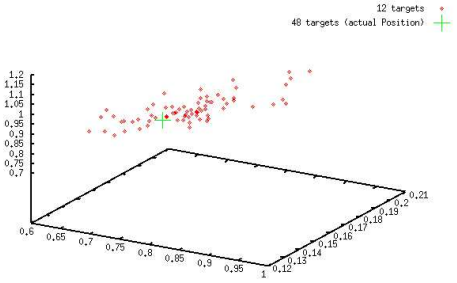

Figure 6.16: 12 targets

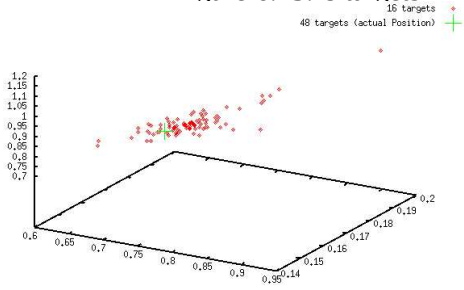

Figure 6.17: 16 targets

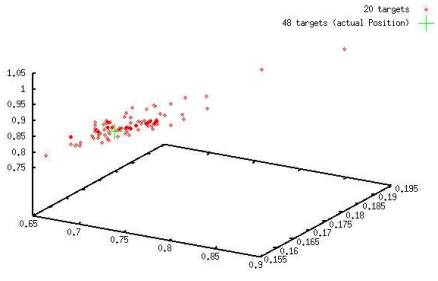

Figure 6.18: 20 targets

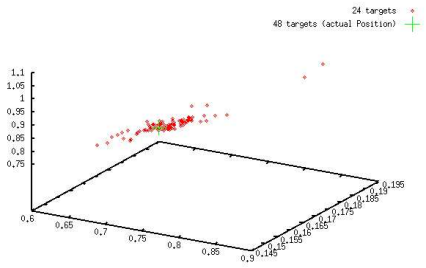

Figure 6.19: 24 targets

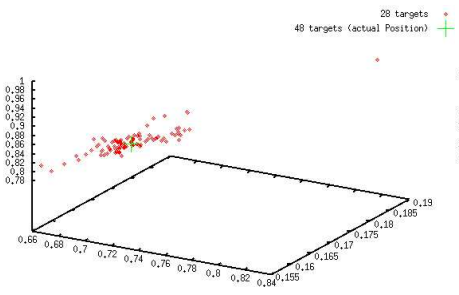

Figure 6.20: 28 targets

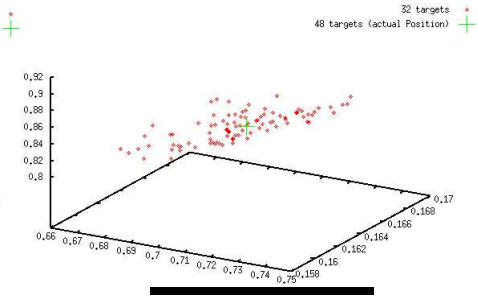

Figure 6.21: 32 targets

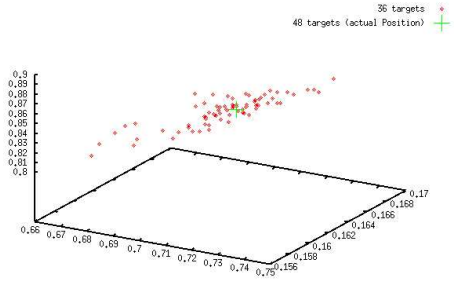

Figure 6.22: 36 targets

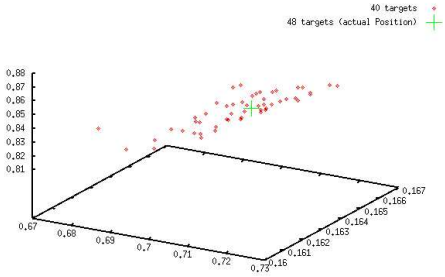

Figure 6.23: 40 targets

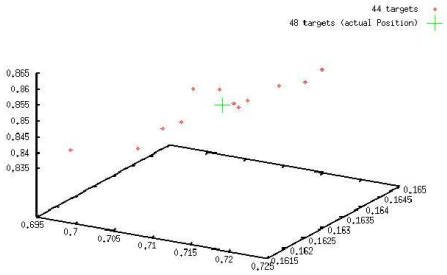

Figure 6.24: 44 targets

other ways so that the comparison would be valid. A single image of the 12 targets of the calibration rig was used. A set of 11 targets can be formed in 12 ways by removing a single target. Similarly a set of 10 targets can be formed in 11×12 ways. To avoid the combinatorial explosion an upper limit of 100, say, is provided to the software test harness and 100 random ways of choosing a particular number of targets is automatically selected. The baseline position to calculate errors is that derived from all 12 targets.

Figures 6.14 to 6.24 show the effect of increasing target number on accuracy using the calibration rig. As the axes scales for these graphs are different, it is difficult to spot the trends in clustering visually. Figure 6.25 combines the graphs for four different target numbers to allow comparison. Trying to combine all the graphs together would be far too visually confusing. Figure 6.26 shows the 2D solutions distinguished from the 3D solutions for the 8 target case. With a single target in view

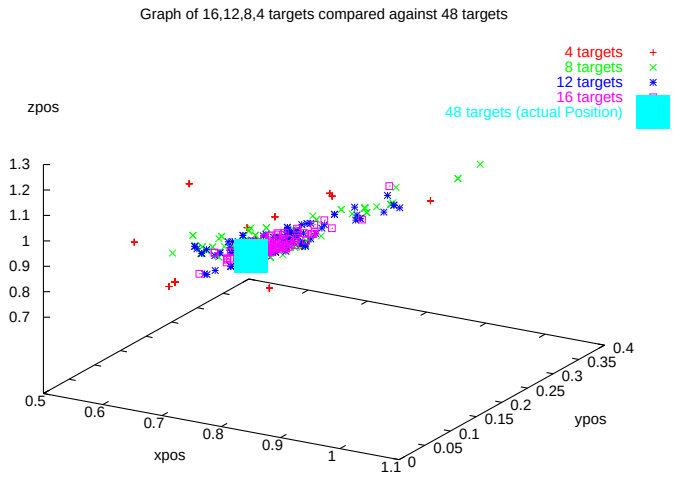

Figure 6.25: Distribution of calculated positions using 16,12,8 and 4 targets

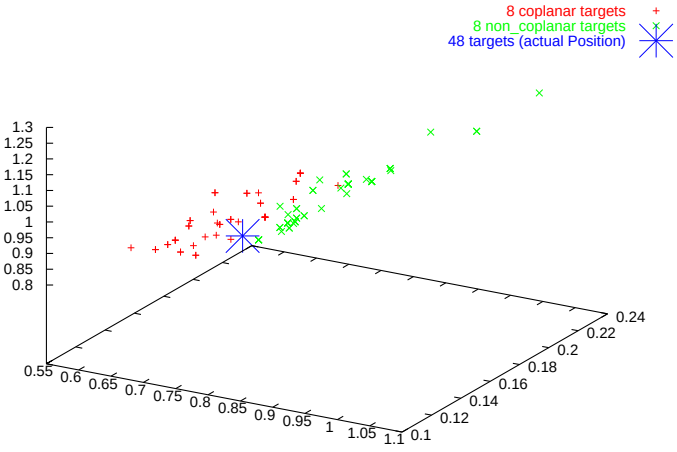

Figure 6.26: Comparison of coplanar and non-coplanar target use

accuracy is around 20 cm, with 11 targets in view accuracy is around 1 cm. The spread of points is observed to be along the camera axis. The positional error decreases with number of targets, but there are diminishing returns after, say, 5 targets. After this stage, the biggest source of errors is elsewhere. A small number of targets is much more likely to produce a wild outlier. Unexpectedly, the outliers were due to the non-coplanar rather than the coplanar method. For the same number of targets, the coplanar results are more tightly clustered and hence are presumably more accurate. It was initially surprising that the 3D results were less accurate as a 3D target set should intuitively be better. Eventually, it was realised that the different computational path in this case was responsible: internal camera parameters are also being determined and this process is prone to noise. The coplanar and non-coplanar results form two distinct clusters.

6.8.6 Varying the Number of Targets: Ceiling Targets

An image of the ceiling was used with 8 identifiable targets, and again the number of targets was varied via a software simulation. This was done in order to compare coplanar and non-coplanar methods using a wider range of target numbers. With the calibration rig there are no sets of 7 coplanar targets. The plot of the coplanar method for the ceiling tiles and the non-coplanar method for the calibration rig, are combined in Figure 6.27. With diminishing target number, the non-coplanar method's error grows more slowly.

6.8.7 Conclusion

The coplanar method is more accurate than the non-coplanar method counter to most people's expectations. The accuracy for the coplanar method may be attributed to the fact that it uses a very accurate precomputed calibration matrix. With the calibration rig, the set-up was found accurate to $\pm$ 1 cm and $\pm 1^{\circ}$ which is adequate for most mid-range AR applications. For the ceiling target set, the accuracy was only $\pm$ 8 cm due to the inaccurate knowledge of the absolute positions of the targets and an initial corrective procedure using least squares analysis would be necessary to rectify this error. However, the next section describes how this error was not a problem in practice.

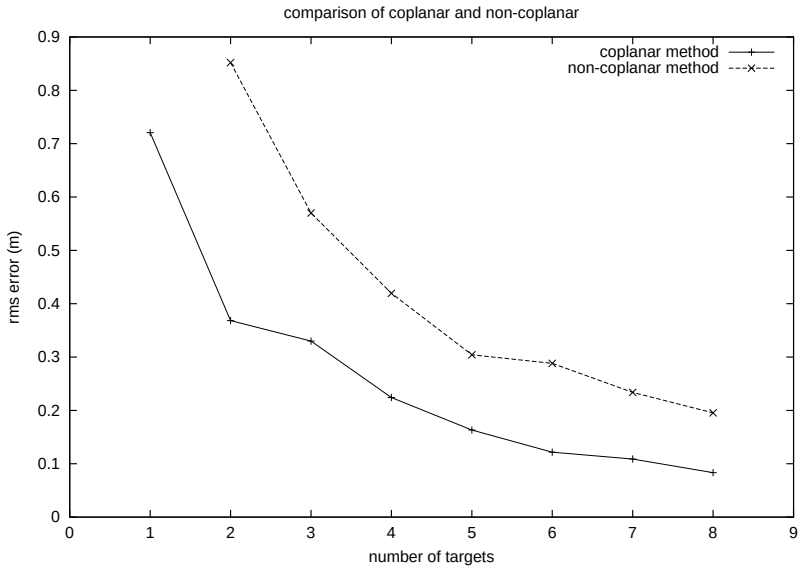

Figure 6.27: rms error of coplanar and non-coplanar methods compared

6.9 Usability and Performance

When the virtual model of the laboratory was superimposed on the real laboratory, registration was sufficiently good that small mistakes in the virtual model could be detected *i.e.* the positional accuracy was better than that of the virtual modelling. The accuracy of the VPS using the ceiling targets is good enough to support an AR application indoors.

When moving around, there was a lag of around four frames before the alignment of the virtual model would catch up with that of the real model. No predictive (Kalman) filtering was attempted. The majority of this latency is attributable to the non-real-time development operating system so little could be done to circumvent this. Subsequent versions of the video grabbing system software are known to have reduced latency.

The AR rig used indoors was rather “string and sealing wax” and the video camera had to be held clamped manually to the VR headset, so runs of more than a few seconds were not practical. Clearly this is not satisfactory in a production setting, and further development will require a proper head-rig such as that used in the University of Washington ARToolkit demonstration system, where a miniature forward-pointing video camera is built into the VR headset, positioned between the user’s eyes [64]. Further, in order to support the 3D graphics a desktop machine rather than the wearable computer has to be used. With cables tethering the user to the desktop, freedom of movement within the laboratory

was limited.

Considerable effort was invested to improve the execution performance of the software and it is believed that the final VPS performance of 4 frames per second on a 266 MHz Pentium is essentially optimal. Both for edge and region processing the code was repeatedly profiled and the time-consuming routines were optimised or re-implemented. The image processing is still the greatest overhead, optimisation only reduced this from around 95% to 60%. Surprisingly, the image processing performance of edge and region processing is almost identical. Edge processing can handle just over 2 full-size PAL frames per second or 8 half-size PAL frames per second. Region processing can handle 4 half-size PAL frames per second, but it has to filter both the original image and the corresponding image which contains squared intensities. The code has subsequently been run on considerably faster machines (*e.g.* 1.4 GHz) with commensurate improvements in performance. However, for consistency all performance figures have been based upon the original development machine. As the performance-critical section of code has been isolated to a single line of software, the application is a prime candidate for porting to different hardware architectures such as FPGAs. The code has subsequently been parallelised within an MPI environment to run on any number of processors within a distributed network.

6.10 Notes on Code Development

The author is indebted to the special effects firm 5D based in London, who provided a pointer towards the Faugeras factorisation method [67] (with sample code) which enabled development of the VPS to continue. The output of this code ported to the VPS environment did not initially seem to make much sense. To investigate the problem, our code for finding $\boldsymbol{P}$ (SVD method) was compared against 5D's code for finding $\boldsymbol{P}$ (eigenvector method). The output agreed to several significant figures, so the problem did not lie here. Finally, it was realised that all that was required was a simple inversion of $\boldsymbol{K}$ obtained by 5D. Special effects firms work with the displacement of objects from the camera, AR works with the offset of the camera in the real world! However, in the meantime a totally independent implementation of the Faugeras method was undertaken, to locate the suspected error in the 5D code. However, results from the two different implementations were again found to tally very closely. This provided useful confidence checking in the correctness of the code. A final implementation of the 3D Faugeras method was produced, based on the findings that came from the exercise of integrating the

resulting matrices with the OpenGL graphics standard. The algorithm for the 2D computational path was developed specially for the VPS. This implementation ensured that $\boldsymbol{K}$ was in a standard format with well-defined semantics.

An API has been written for the VPS and the code has been made available in the Internet. There is a considerable paucity of public domain target detection code, and the API has been designed so that the target detection or the geometric processing of the VPS can be used in isolation. Considerable effort was placed in ruggedising, productising and rationalising the VPS, because it was felt the functionality of the VPS would be of considerable use to other research groups. An exemplar application called "vps" has been written using the VPS library, which performs the majority of tasks that most users require. The VPS API is documented in Appendix G, a couple of undergraduates providing much of this effort [71] [72].

6.11 Supplementary Applications

The vision system developed is a generic position and orientation determining engine and it has been used in a variety of applications other the "subjective" AR one under discussion.

6.11.1 Shared Virtual World

By walking around a room a human user (Figure 6.28) can control the position and orientation of an *avatar* in a virtual representation of the same room (Figure 6.29). The video camera is strapped pointing upwards to the user's head, while targets are stuck to the ceiling. The demonstration that has been developed allows any number of conventional terminal-based users and video-equipped ambulatory ones to interact in a shared virtual world.

6.11.2 Video Joystick

Instead of working with a moving camera and fixed targets, it is possible to work with moving targets and a fixed camera. A 10 cm cube has distinguishable targets pasted on each of its 6 faces. The position and orientation of the cube is measured by the vision system. These are used to control either the position and orientation of an object in an on-screen virtual world or the position and orientation of the viewport into the virtual world. The cube acts as an input device supplying 6 numbers. There are no cables to interfere with manipulation of the cube.

Figure 6.28: Human controlling avatar's position

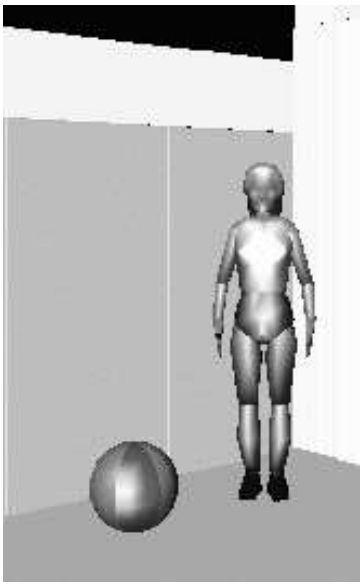

Figure 6.29: Avatar's position being controlled

Operation of the cube is a completely transparent and almost the ideal way of moving an object in the scene, as the real cube and virtual object are locked into a one-to-one correspondence (Figures 6.30 and 6.32). Figures 6.31 and 6.33 shows the detected regions, revealing that two and three detected targets respectively are sufficient for good registration. The video joystick has user guidance (by way of visual feedback) so the VPS automatically provides acceptable positional accuracy, as visual locking is achieved.

6.12 Conclusions

The video-based positioning system described in this chapter has proven robust, reliable and sufficiently accurate for the applications described. All requisite processing, including the vision processing described in the previous chapter can be done using inexpensive PC class hardware. Anyone with a monochrome PostScript printer and access to a digital video stream will be able to use the system, as the software has been placed in the public domain.

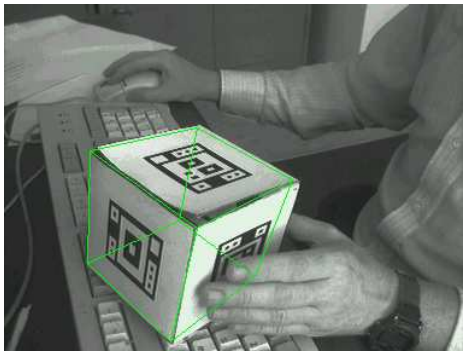

Figure 6.30:
Wireframe cube augmenting scene:
2 faces visible

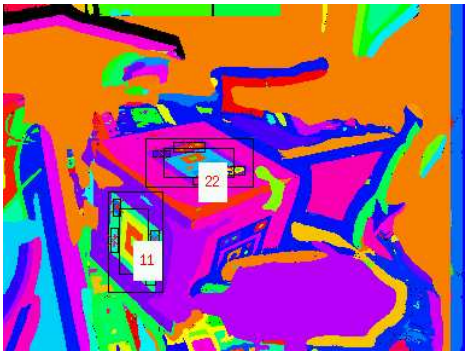

Figure 6.31:
Region processing after painting:
2 faces visible

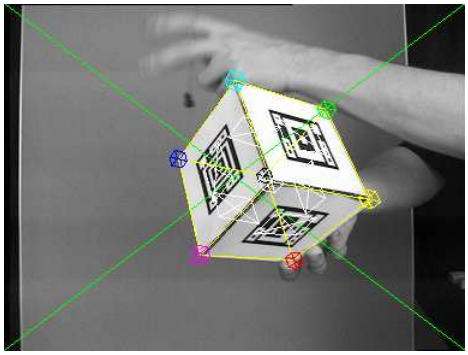

Figure 6.32:
Wireframe cube augmenting scene:
3 faces visible

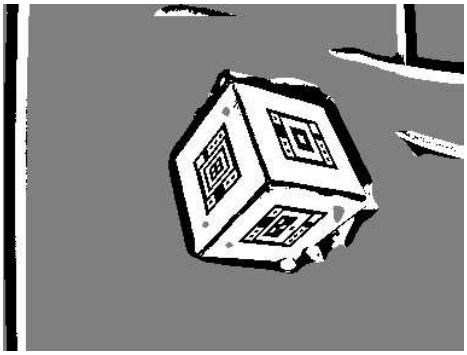

Figure 6.33:
Region processing before painting:
3 faces visible

Chapter 7

Hybrid Sensor

7.1 Introduction

The individual positioning technologies that are useful for Augmented Reality have their drawbacks. This is certainly true of GPS and computer vision which are the two principal techniques that have been investigated in this thesis. Conventional GPS algorithms require four satellites to be in line of sight, so the technology does not work indoors or outdoors where too much of the sky is obscured. The consequent inability of GPS to work in built-up urban areas (so-called "urban canyons") is its biggest flaw. Vision-based positioning requires targets to be in view and, if too few targets are in the scene, the system again breaks down. An obvious solution is to combine a number of different technologies to overcome the individual deficiencies of each. The composite system may be presented as a single *hybrid* sensor of synergistically superior specification. This chapter discusses the use of hybrid sensors within AR, with particular emphasis on GPS, and presents some practical work using a micro-machine accelerometer which has the potential for working as one component within an AR hybrid sensor.

7.2 Hybrid Sensor

A hybrid sensor requires two attributes to be effective:

1. A genuinely complementary set of component sensors: Indeed, even a technology which is useless on its own for positioning may have sufficiently good complementary properties to be a valuable component of a hybrid sensor.

2. The appropriate algorithm to combine the component readings: The fully general case is a research topic. However, the algorithms for particular technological combinations are well represented in the literature, and are found within commercially available products.

Consider some particular examples which illustrate appropriate complementarity:

Short Term Relative *vs.* **Long Term Absolute** Sensors which measure acceleration, called Inertial Measurement Units (IMUs), can act only as relative position sensors. Measurements are relative to the initial inertial frame of reference. Furthermore, because a position determination requires a double integration, the positional error will grow as the square of time. Over long timescales IMUs are useless for determining position but over shorter timescales they can be quite accurate. Contrast this with a computer vision system which produces an absolute position fix but only occasionally, due to either processing time, processing latency or to infrequently visibility of targets. The IMU is ideally suited to "in-between" the measurements of the vision system. Another suitable combination that would work in this fashion is GPS and an IMU.

Higher Relative Accuracy *vs.* **Lower Absolute Accuracy** The GPS systems developed which used both carrier phase and code phase measurements were essentially hybrid sensors. The carrier phase is a relative range measurement of high accuracy and the code phase is an absolute range measurement of lower accuracy. The two were successfully combined in a number of different ways, to produce an absolute range measurement of high accuracy.

7.3 Hybrid Algorithms

Provided either an automatic or manual initial registration occurs with an AR system, then only relative but not absolute accuracy is required. However, perhaps this distinction between initialisation and non-initialisation modalities or between absolute and relative measurements is artificial. What all measurements provide are constraints on the actual position. The most likely solution that satisfies all these constraints is sought, and a generic hybridisation algorithm will make no distinction of category. The conclusion is that a good hybridisation algorithm will automatically handle any "initialisation" transparently in an AR system and therefore should be sought out.

One complication is that the readings from the different technologies occur at different times and at different rates. So at any one instant when a position solution is requested, then the older readings will be less relevant. The constraint supplied by a single reading slackens with time. Similarly, one

does not just simply need to use the last reading from any one technology but one can use a time series of readings to improve accuracy *e.g.* Kalman filtering [73]. Further filtering can be used, once values from the different technologies are combined.

Given inevitable latency in any AR pipeline, should one use prediction in determining a position fix *i.e.* provide a position fix for the time in the future when the composite image is expected to be presented to the user? There are some indications that this is effective [27]; however, in the systems produced for this project the latency is not modelled and the position is determined at the time of request, which is generally the time of reading of the fastest-turnaround technology. AR is an interactive discipline so post-processing (using future as well as past measurements) is not valid.

How are the component technologies to be combined? Suppose the first technology returns position P_1 and the second technology P_2. Is the optimal solution some weighted sum of these two positions? There are a couple of reasons why this is not the appropriate level to combine results:

1. Some of the component technologies involved may not even return a position, just some kind of positional constraint. For example, if a GPS system has only 3 satellites in view it cannot provide a position fix (the standard algorithm needs 4 satellites). However, it would be unwise not to use the three range measurements in some way.

2. It is a closer mathematical modelling of the situation to retain each component measurement and associated uncertainty. This is also better science as errors can be propagated. Some measurements have complex technology-dependent error characteristics that would otherwise be lost. For example, the GPS carrier phase range is only a measurement of the actual range modulo an unknown whole number of wavelengths. It would be difficult to preserve multiple instances of this special quantised quality of uncertainty in a single position fix and single associated error.

Overall, it is clear that all component mathematical constraints from all component technologies should be used to produce a global solution to the position fix problem. The general solution to a set of time dependent constraints is a difficult problem. The majority of such algorithms in the literature stay within the domain of a single technology (such as GPS) or two technologies (such as GPS and inertial systems) but do not address an arbitrary number of technologies. Even in the case of a single technology, the solution techniques can be complex *e.g.* the LAMBDA method of Chapter 3.

7.4 Technologies

Inertial sensors and other "rag-bag" technologies such as odometers and electronic compasses are ideal components for hybrid sensors. Mechanical gyroscopes are IMUs that have been traditionally used in high-end navigation applications, but these are expensive and bulky. However, low cost and small form-factor gyroscopes are now available *e.g.* the piezoelectric vibrating gyroscope. Even newer gyroscopic technology is being micro-machined into silicon, and there is good promise of a gyroscope on a chip. Inexpensive micro-machined accelerometers in silicon have appeared in the market-place in recent years. The main application of these accelerometers to date has been to inflate car air-bags on accident impact. However, these devices are capable of measuring quite accurately and their use in a quantitative manner rather than a qualitative one is fairly novel.

Overall the trend is clear. The type of navigation technology that was once only available on board aircraft is becoming small and cheap enough to be worn by the individual. Off-the-shelf integrated GPS/INS (INertial System) units are currently available for the aviation industry. Integrated GPS/INS systems have been demonstrated to be effective for accurate (autonomous) vehicle tracking [74]. The technique is even good enough to detect GPS multi-path error: here measured ranges to the satellites are inaccurate because local environmental conditions cause reflected signals to have greater intensities than direct ones. Will we ever be able to buy an off-the-shelf personal navigation module based upon hybrid sensor techniques? The questions are purely technological:

1. Are the miniaturised navigation transducers capable of suitable accuracy and performance for the characteristics of human motion? Some of the military grade gyroscopes will only cope with angular velocities of say 60° per second. A human is far more maneuverable than an aircraft, but at least accelerations of the human body are known to be under $2g$.

2. Will the commercially valuable hybridisation algorithms and software become cheap enough for personal use or public domain? No such public domain code is available. A desirable outcome of this work would have been such software, in the same way that the differential GPS codes developed have been made public domain.

The VR headset mentioned in Appendix J is an example of the move towards such technologies. The headset is instrumented to provide head orientation information: yaw is provided by a Hall Effect sensor (essentially an electronic compass which gives a bearing); pitch and roll are each provided by

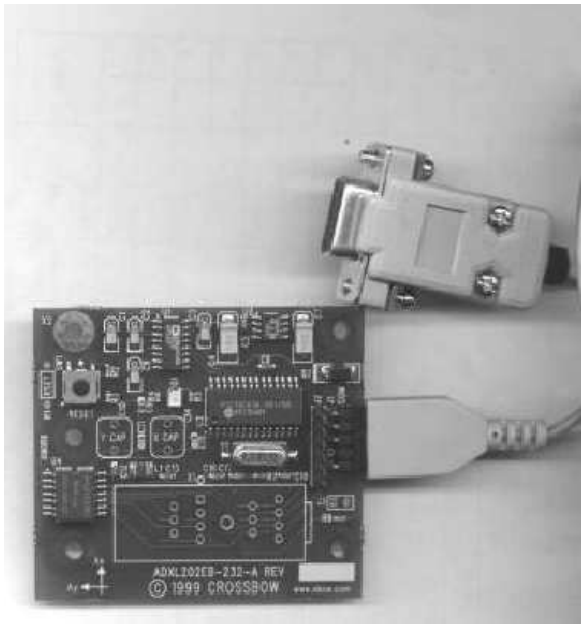

Figure 7.1: ADXL202 evaluation board

a dedicated mercury tilt sensor. As the angle of the mercury surface changes within a sealed glass phial, the resistance between built-in electrodes changes accordingly. The accuracy is crude (typically being out by a number of degrees) and so unsuitable for AR, yet the feedback made possible, when say viewing a virtual 3D scene, is effective.

7.5 Experimentation Using an ADXL202 IMU

The micro-machined accelerometer technology was identified as worthy of investigation for potential hybridisation with the GPS system developed due to its low cost and small form factor. A development kit for the ADXL202 micro-accelerometer was purchased. The ADXL202 from Analogue Devices is a dual axis accelerometer measuring accelerations up to 2g within the plane of the chip. The evaluation board (Figure 7.1) interfaces the IMU to an RS232 interface, allowing a host computer to read both X and Y accelerations at up to 275 Hz with 16-bit accuracy. The sensors themselves are micro-machined fingers of silicon, that flex under the forces to be measured. The 2-axis tilt sensing of the ADXL202 responds faster than conventional electrolytic, mercury or thermal sensors and indeed there is a potential use here for the IMU as a novel computer peripheral within the AR domain. However, we are mainly concerned in this chapter with the positional capability of the device. Note that tilt is

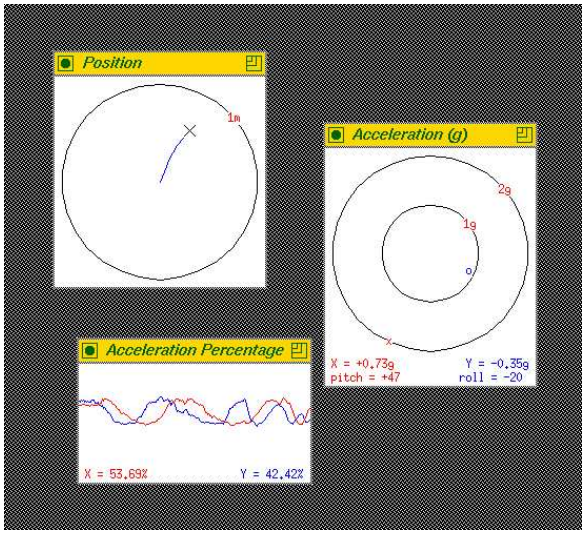

Figure 7.2: Accelerometer software

indistinguishable from acceleration, due to the ever-present force of gravity. It is similarly impossible to distinguish rotation of the board from linear acceleration, though it would probably be possible to get around this in the planar case by attaching two sensors some distance apart on a rigid body. Differential acceleration between the two sensors would thus represent rotation. An alternative way of distinguishing linear acceleration and rotation would be the use of a gyroscope.

For initial investigation of the ADXL202 as a positional device some simplifying assumptions are made. Firstly, the evaluation board is assumed to remain flat and unrotated; a full IMU-based system would have to take into account coordinate transformations and use an orientation technology. Secondly, the effect of the earth's rotation is assumed to negligible. The centrifugal and Coriolis forces can be neglected for the short time-scales of the experiments, but should be modelled if longer time intervals are being studied.

7.5.1 Software Produced

Software was written under Linux to talk to the evaluation board, as only Windows software was provided with the product. Graphics were written using the X-Simple library (described in Appendix H), that had been developed earlier. The graphical display of three windows is shown in Figure 7.2. This consists of:

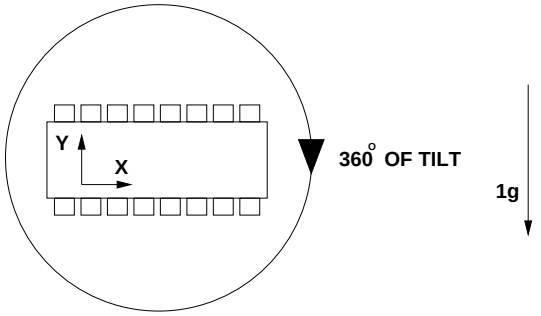

Figure 7.3: Measuring tilt

Percentage traces for the X and Y accelerations. The evaluation board presents acceleration as a percentage. This is the ratio of pulse width to period of the square waveforms that the ADXL202 produces for both X and Y accelerations.

Acceleration in g. The percentage accelerations are turned into real physical units and plotted on a 2D display. The unit used is g (the acceleration due to gravity) which is approximately 9.82 ms^{-2}. If the evaluation board is stationary, this plot will stay within the $1g$ circle, and the 2D display then acts as a circular spirit level. The equivalent pitch and roll angles are displayed. The ADXL202 can also be used to measure a full 360 degrees of tilt: the evaluation board must be oriented vertically and rotated about its perpendicular axis (Figure 7.3). A cross travels around the 2g circle to represent the angle of tilt.

Position plot. The acceleration is integrated twice to give a position which is plotted in 2D. Position (and velocity) will inevitably zoom off to infinity, so how can this be treated in a display of finite size? The solution is to reset the velocity and position to zero, once the position falls outside the plotting area.

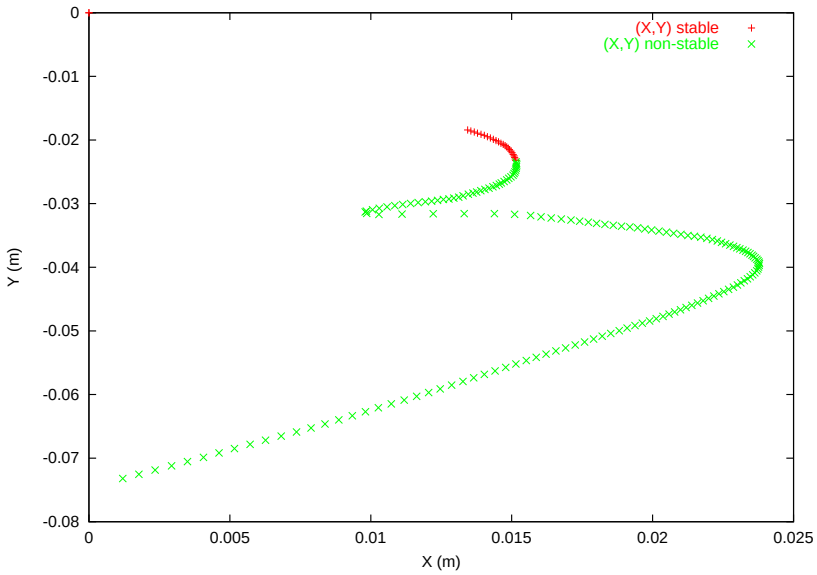

Figure 7.4: Accelerometer positional plot

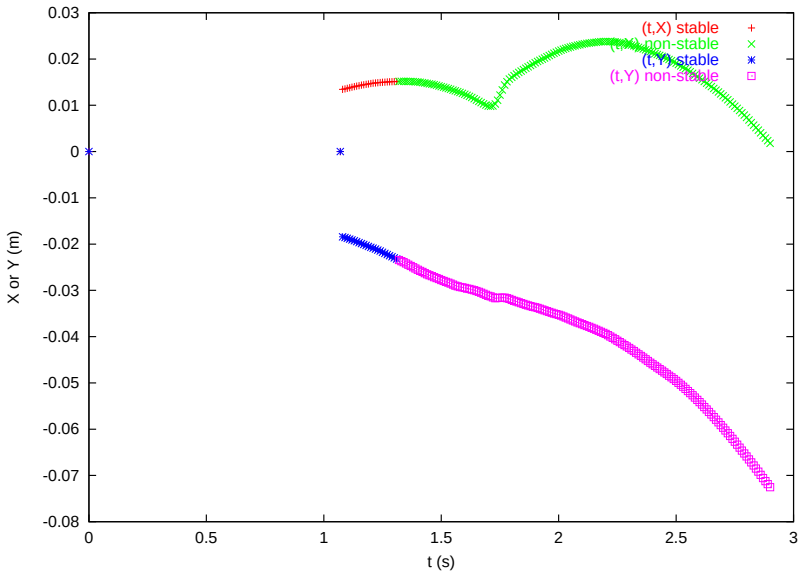

Figure 7.5: Accelerometer (X, t) and (Y, t) plot

7.5.2 Results

The position plotting window gives an intuitive feel for the accelerometer as a positioning device. A simple experiment is to trace the accelerometer board quickly over some cursive writing or some such pattern, to see if this pattern is visible within the position plot window. This is only partially successful, and needs to be done exceptionally quickly and immediately after the position and velocity have been reset to zero. This shows that the time window of opportunity for using the accelerometer in isolation as a positioning device is very short indeed. For example, Figure 7.5 represents the accelerometer being rotated rapidly in small clockwise circles of a couple of centimetres in diameter. Due to the rapid positional drift, not a single complete closed circle is drawn. One could perhaps say that the first quadrant of the first turn shows some spatial coherence *i.e.* the accelerometer is stable as a positional device for this small period for the scale of movement involved. Looking at the individual plots of X and Y against t in Figure 7.5 shows that the quadrant only represents 0.2 seconds. The temporal plot is possible because the software timestamps each acceleration measurement as best it can.

With the accelerometer stationary and level, the position display drifts to 1m after around 5 seconds. The rate of drift depends not so much on the accuracy of the accelerometer but on the accuracy of the previously determined zero-point for the accelerometer. Due to inaccuracies in the measured $0g$ reading, the drift will almost always be biased in one particular direction. Unfortunately, this zero point exhibits some hysteresis *i.e.* if the accelerometer is rapidly moved then the zero point is **not** quite the same as before. The best that can be done is to set up a fairly good average zero point for the device in use. Different versions of the same model of accelerometer were observed to have radically different zero points. Initially the calibration of the device for each axis was carried out as follows:

1. orient the evaluation board at $+90$ degrees for the axis in question. Take reading p_1.
2. orient the evaluation board at -90 degrees for the axis in question. Take reading p_2.
3. determine zero point as $\frac{p_1+p_2}{2}$ and scale factor to convert from percentage to g as $\frac{p_1-p_2}{2}$.

However, it was eventually realised that the zero point is so critical that it is best to measure this directly, by making the board horizontal. The scale factor is calculated as before.

A second positional plot is shown in Figure 7.6 where the accelerometer has traced out four clockwise circles. The starting and finish point of the movement were deliberately engineered to be

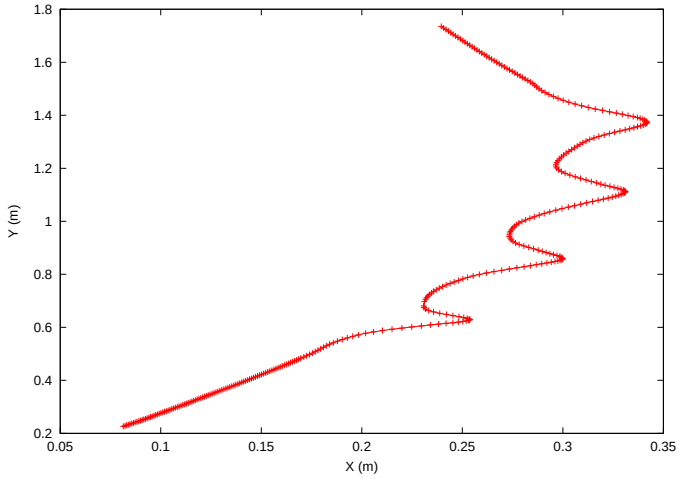

Figure 7.6: Four circle positional plot : uncorrected

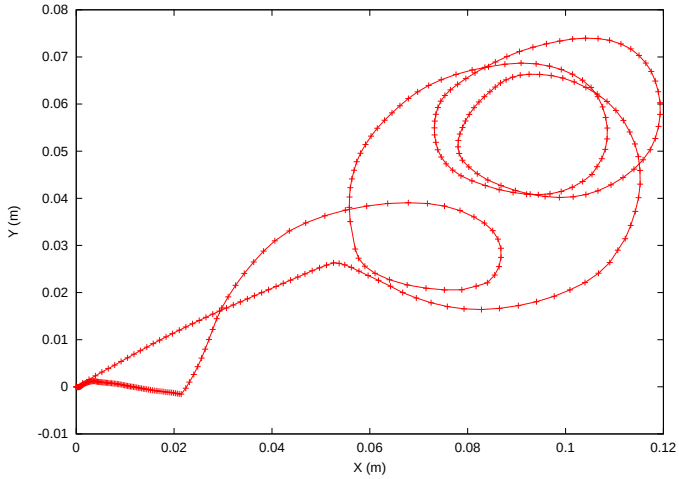

Figure 7.7: Four circle positional plot : corrected

the same. This time, however, the data have had some additional post-processing to find out a more accurate zero acceleration point. Using an awk script that employs a gradient descent method, the zero points for both X and Y accelerometer readings were chosen to minimise the computed distance between the starting and finishing points. On the corrected plot, Figure 7.7, the tracing of the four circles can clearly be discerned. The shorter of the straight line segments represents the accelerometer at rest before the movement. The longer of the straight line segments represents the accelerometer at rest after the movement. This simple zero-point correction technique has extended the temporal window of usefulness for the accelerometer as a positional device. The plot represents a period of 3 seconds. However, the zooming off to infinity with time is still apparent in the velocity information. The distance between the points on the straight line segment before the movement are much closer together than the points on the straight line segment after the movement. Another reported cheap feedback trick is to assume that the average acceleration over a run is the downwards acceleration due to gravity, but the author is rather sceptical about this.

In practice, a GPS device could provide the corrective information for a sliding window of accelerometer data. In this case the corrective input was simply knowing the start and finish points were the same and the processing was batch-oriented rather than dynamic. Although this unrealistic correction is far short of a full hybridisation technique, it does illustrate how additional knowledge from another source can compensate effectively for the deficiencies of a particular technology. A more sophisticated sliding window scheme could compensate for a time varying zero acceleration point, and full hybridisation could address lower order effects.

With all graphic displays on, the fastest the system can read acceleration data is 33 Hz. With all graphics switched off the fastest the system operates is 100 Hz, despite the fact the RS232 was running at the maximum 38400 baud. Even taking the application from a 266MHz PC to an 800MHz PC makes no difference whatsoever to these performance figures! The conclusion is that the application is bandwidth limited by practical RS232 performance. The theoretical performance can be estimated:

$$f_{theoretical} = \frac{38400 \text{ baud}}{10 \text{ bits per byte} \times 5 \text{ bytes per cycle}} \approx 768\text{Hz}$$

The bi-directional nature of the traffic is bound to introduce overheads, but the achievable performance falls well below even that specified. An attempt was made to use buffered I/O instead of unbuffered I/O to increase the performance. However, although the buffered write worked, the buffered read could **not** be made to work. A slight alteration of the code as shown in Table 7.1 was found to increase

`while ( TRUE )`
`{`
`  send acceleration request`
`  read system clock`
`  read acceleration data`
`  computation and display`
`}`
	`send acceleration request`
	`while ( TRUE )`
	`{`
	`  read system clock`
	`  read acceleration data`
	`  send acceleration request`
	`  computation and display`
	`}`

Original Code: runs at 25 Hz	Rearranged Code: runs at 33 Hz

Table 7.1: Performance enhancing code rearrangement

performance marginally probably because of the increased overlap of main CPU and evaluation board activity, but this only worked when graphics were in use. The system clock on the PC is read to timestamp the acceleration readings, so the appropriate dt can be used for integration purposes. In fact, it is impossible to guarantee the acceleration data and the clock data have been read at the same time: the best guess was to read the clock just before the receipt of the acceleration result.

7.5.3 Conclusions

Due to the vagaries of timings both in data transmission and in system-induced latencies it is clear that the accelerometer readings should ideally be time-stamped at source. This is not to say that a real-time clock should be built into the hardware, simply that a 16-bit timer (say) can be used to give a more accurate handle on the appropriate dt to use for integration. There is an argument also that the PIC (Peripheral Interface Controller) or other device looking after the accelerometer should perform the integration (just two multiplies and two additions) as a dedicated unit can more frequently sample the transducer than a general purpose operating system. However, this also requires that the main CPU can reset velocity and position values on the PIC, increasing the complexity of the protocol. The maximum rate at which acceleration readings could be made using a PC was surprisingly low at just 100 Hz, irrespective of the processor speed. Intuitively, the more frequently acceleration can be measured and the more accurate the corresponding time measurement, the more accurate the final position calculated will be. There will be a critical time period, before which these “integration” errors are dominant, and after which the drift error is dominant.

The time interval for which the accelerometer is a useful position device on its own is very short: of the order of 0.1 s. To hybridise this realistically with a GPS unit (say) probably requires either the GPS unit to take very frequent position measurements (*e.g.* the 10 Hz of the AllStar unit would probably have been suitable) or the error of the GPS unit to be much larger than it currently is (*e.g.* pre-SA switch off). These comments have been made without a proper quantitative analysis, and it is an open question whether the particular IMU technology and GPS units investigated could hybridise successfully. However, it has been demonstrated how even simple processing can greatly extend the useful time window of the IMU to 3 seconds or so.

Overall, the ADXL202 is regrettably not suitable as a hybrid component in a GPS system: it is a 2D rather than a 3D device and the absolute prohibition on rotation necessary to obtain sensible data is not practical to apply outdoors for ranges of movement that are significant for GPS operation. Although the ADXL202 is a "toy" technology, the numerical results were sufficiently encouraging that the next step would have been to purchase a compact full six-axis accelerometer, that can distinguish between rotation and lateral acceleration.

As working differential GPS equipment was only obtained within the last three months of the experimental phase of the project this period was largely spent commissioning, correcting and verifying the GPS software, and then investigating particular techniques for high accuracy GPS. This two year delay in the arrival of working differential GPS equipment prevented the investigation of sensor hybridisation, an aspect of the project which was ironically awaited with much keenness. Time constraints prevented further investigations with compact six-axis accelerometers.

7.5.4 Other Uses for the ADXL202

As a standalone device, the ADXL202 accelerometer is actually more effective at measuring angle than position. In fact, when the board works in the mode of measuring a full 360 degree tilt angle, the behaviour of the device is superb being responsive, stable and (to a cursory examination) accurate.

Mounting two ADXL202 chips perpendicular to one another would provide limited 3D angular orientation (*i.e.* pitch and roll, but not yaw) providing novel peripheral control not just for 3D orientation-based tasks. Placing a number (*i.e.* more than 4) of these accelerometers in a non-coplanar configuration on a rigid body would start to allow acceleration and rotation to be distinguished, and through redundancy would provide even more accurate angular orientations.

A technical note on the ADXL202 produced by AMD [44] also suggests that it may be possible

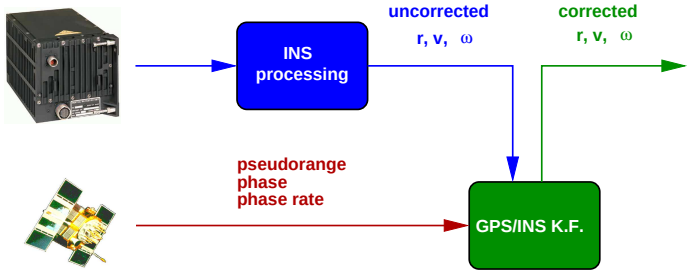

Figure 7.8: Open loop filtering

to track an individual by placing accelerometers (and electronic compasses) in their shoes. However, the note does **not** produce positioning results, indicating either that the challenge is harder than the optimistic tone of the article would suggest **or** that the system has not actually been built **or** most likely both!

7.6 GPS Hybridisation

7.6.1 Existing INS/GPS Hybridisation

Hybrid INS and GPS sensors are available commercially from manufacturers of INS equipment. Illustrative of the methods employed is software from the University of Calgary called KINGSPAD (KINematic Geodetic System for Positions and Attitude Determination). Cycle slips in GPS data can be detected and removed using the high short-term accuracy of the INS velocity. The accurate differential GPS position frequently updates the inertial system. This calibration provides a major reduction in INS orientation errors. Two different approaches to filtering are possible in the hybridisation:

Centralised Filter Approach A common state vector models both GPS and INS error. INS is used to calculate the reference trajectory while GPS updates the solution and estimates the state vector. This is shown in Figure 7.8.

De-Centralised Filter Approach The GPS data is Kalman filtered to obtain position and velocity estimates. There are used as quasi-observables to update the INS Kalman Filter. The GPS data is checked for cycle slips This is shown in Figure 7.9. This approach is considered to have the advantage in terms of data integrity and speed.

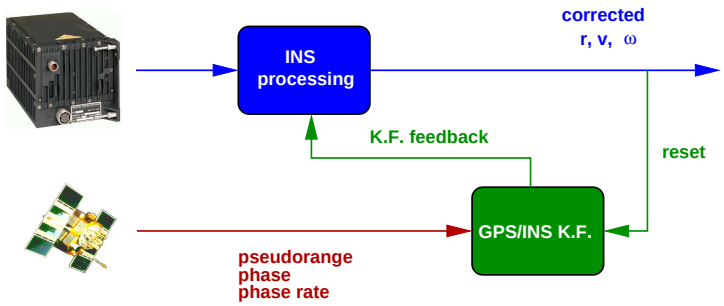

Figure 7.9: Closed loop filtering

It is an open question, post SA switch-off, whether a standalone GPS receiver on its own could complement an IMU. The hybridising capability of differential GPS may be its saving grace, given that single receiver GPS is now good enough for most applications that demanded differential GPS in the past.

7.6.2 Proposed Hybridisation for GPS Failure

There are other obvious technologies that can be used for GPS hybridisation *e.g.* the odometer in a car, an electronic compass, an electronic altimeter and cell phone ranging. In fact there is a GPS receiver on the market which features both a compass and altimeter, but these additional peripherals are not used in a hybrid way. In a non-trivial manner, a subsidiary hybrid sensor component may come into its own, when the principal component technology fails. A case of this would be where less than 4 satellites are available for GPS. Let us consider the cases:

3 visible satellites. The only reason a fourth satellite is essential is to correct the low accuracy clock on the GPS receiver. However, the GPS fixes obtained with 4 or more satellites previously could be use to model the drift of the cheap clock, and make corrections into the future.

2 visible satellites. Two range spheres intersect in a circle. However, provided the altitude of the user has not changed considerably since the last position fix, then the assumption of constant height gives another sphere of intersection to yield a point. Use the receiver clock correction described above.

1 visible satellite. A single range from a satellite is not a lot to go on. However, an assumption of a constant altitude and a single range gives a circle of probability. A constant velocity could be assumed and a guess near the last known position could be made. Knowledge that the user is

on foot could provide some useful constraints. Perhaps an accelerometer built into the GPS unit could detect that the user is walking?

The constant altitude assumption could be replaced by the reading from an altimeter or a geographic database, so there is a blurred distinction whether back-up information is guessed, taken from a sensor, or fetched. Even though traditional GPS receivers give up when less than 4 satellites are visible, it is clear that much can still be done to determine position. While there may be loss of accuracy, using a variety of techniques and the devices mentioned above this may not be catastrophic. It is preferable to provide a location with a quality of service indicator, rather than no position at all as happens at present. In tests in the streets of London, it was typically found that just two satellites were within line of sight, but significantly it was very rare that the number dropped below two. There has been very little work on determining the most likely position of a user, given a rag-bag set of input measurements. Whereas, a rag-bag set of sensors would currently be a rather bulky proposition, there is plenty of scope for having these micro-machined onto a small area of silicon.

7.7 Conclusions

Sensor hybridisation is undoubtedly a useful technique for AR because it exploits, in terms of accuracy or reliability or both, the redundancy present in a collection of positioning technologies. Currently INS and GPS hybridisation is effectively used for aircraft and vehicle navigation, and it is only a matter of time before such technology can be transferred to the human body. For example, a hybridised combination of GPS and a full six axis inertial navigation system is employed by the RT3000 vehicle navigation system from Oxford Technical Solutions [74]. This provides orientation to a tenth of a degree and position to within 2 cm at 100 Hz. When such a system (currently weighing in at 1.7 kg) can be miniaturised at low cost then personal AR becomes feasible.

Generic hybridisation is both under-used and under-explored. There is clear scope for GPS ranging information as a hybrid component (even in conjunction with historic data from the same unit) when too few satellites are in line of sight causing conventional dedicated GPS algorithms to fail. The experimental work carried out in this chapter is preparatory, and due to time constraints did not progress to full GPS/INS hybridisation with a full six-axis accelerometer. However, the use and potential of a small and inexpensive micro-machined accelerometer was illustrated.

Chapter 8

Visualisation Architecture

8.1 Introduction

For both video-overlay and optical-overlay AR, the choice of the system which produces the graphical output is key. High-quality rendering systems are commonly available, but fitting such a system within an AR framework places additional demands, which are surprisingly difficult to satisfy in practice. This chapter describes how the following challenges of visualisation for AR were addressed:

- real-time performance
- low latency
- control of viewpoint from novel position measuring devices
- support for 3D display devices (*e.g.* a virtual reality headset)
- compatibility with format of 3D models

8.2 Visualisation Data

A starting point for the visualisation system was the presence of pre-existing Virtual Reality Markup Language (VRML) models of Roman buildings that had once stood on sites around Colchester. These VRML models are produced by Tcl scripts, written by Dr. Christine Clark through collaboration with local Colchester archæologists [75]. Figures A.1 and A.2 show views of the temple complex at Gosbecks Archæological Park.

VRML has been hyped as the *de facto* standard for interactive 3D graphics, and promises a suitably future-proof, Web-transparent representation. VRML2 is the first version of VRML with interactivity. Code is associated with the objects within a scene. Events within the scene will trigger execution of this code. In addition to using pure VRML2, object code can also be written in Java or JavaScript. Indeed, the integration of Java and VRML was one of the reasons why Java was the original candidate language for the entire AR system.

However, initial experience of Java revealed critical bugs in the standard class methods and exceptionally poor performance, so the language was abandoned. The majority of the time porting a test application to Java, was spent working around the problems in the Java libraries, and it was clear that Java, at this stage, was too immature for serious AR work. Subsequently, Java has become more stable and even has a Java3D API, though the author doubts whether the missing facilities in the language have yet been supplied (in particular certain RasterOp functions were missing from the class methods specifications). In short, Java missed its window of opportunity as a development language for this work. Initial good intentions of obtaining high portability though Java were shelved. Ironically, aside from the platform-independence of Java bytecode, it must be remarked that Java source code was found to be considerably less portable between machines than 'C' code.

There are serious omissions in VRML as a language. There is no loop construct. A colonnade is a natural consequence of wrapping a loop round a column object, but VRML insists on each column object being instantiated individually within a VRML code listing! The result alone is that VRML is considerably less compact and tractable than it could be. This fact makes the use of an intermediate scripting language almost necessary for algorithmically generated VRML, immediately invalidating VRML as the primary representation for 3D objects.

8.3 Visualisation Hardware

The i-glasses from Virtual I/O Systems is the VR headset being used for the AR application. Not only can the headset present stereographic images to the user, but it can measure the angular orientation of the user's head. No software support was supplied with the i-glasses, merely some Windows-based demonstration programs. In consequence, driver software had to be developed for the Linux operating system. The hardware for the display, the hardware for the graphics rendering, the input sensors and how to drive the display are discussed separately in the following sections.

8.3.1 Display Hardware

The i-glasses are driven from a standard VGA video signal of 640 pixels horizontally by 480 pixels vertically at a 60 Hz refresh rate, However, there is no information even in the "Developer Kit" pamphlet on the exact video timings required. Experimentation revealed that some latitude was permissible in the nature of the input signal. The actual resolution of the LCD screens built into the headset is undoubtedly less than VGA, though there is no consensus in the literature as to what this resolution exactly is. The specification document from Virtual I/O talks about a buffer of 320×400 distributed scan line by scan line to two 320×200 buffers - one for each eye ($320 \times 200 \times 3$ (for Red, Green and Blue) = 192000 pixels/eye). On the other hand "The Virtual Reality Home Brewer's Handbook" mentions a resolution for each eye of 263×230 ($263 \times 230 \times 3$ (for R,B,G) = 181470 pixels/eye). With yet a third angle on the matter, the online specification for the product mentions $180,000$ pixels for each of the two LCD screens. Whatever the resolution, the headset display, though crude in quality by modern standards, is actually more than adequate for prototype work and is far from being the weak link in the system. Ultimately though, tourists will demand a higher-quality visualisation system. However, the model itself will have to be rendered in a more sophisticated manner (say using texture mapping) before any increase in the resolution of the display is worthwhile. In order to display a stereographic image on the i-glasses, two conditions have to be met:

1. video signal meets appropriate requirement (*e.g.* 640×480 pixels @ 60 Hz)
2. left/right eye image is on odd/even scan lines of screen respectively

8.3.2 Driving the Headset Display

Changing the Screen Resolution under Linux

Test runs using the VR headset are necessarily performed at a screen resolution of 640×480. A code development environment normally runs on a high resolution monitor displaying 1280×1024 pixels. An initial development cycle thus required closing down the windowing system; reconfiguring it; running it at a different resolution for testing and then following the reverse route. Fortunately under X-Windows it is possible to work with a virtual screen larger than the physical screen. At any one time a particular sub-window onto the virtual screen is displayed, and by scrolling the graphics cursor to the edge of the screen this sub-window can be caused to pan about. The physical resolution of the screen

dot clock rate	Result
28 MHz	fails
27 MHz	fails
26 MHz	works
25.175 MHz	works

Table 8.1: Viable dot clock rates for i-glasses

may be altered using the key combinations <ALT>-<CNT>-<+> and <ALT>-<CNT>-<-> to cycle up and down respectively though the preconfigured resolutions. This enables switching between development and test modes by only using a few keystrokes — which proved much more satisfactory.

Unfortunately, the configuration programs (xf86config, XF86Setup and Xconfigurator — there may be more) assume that one wishes to drive one's monitor, for any particular screen resolution, at the highest possible refresh rate. The result is that if one configures for a monitor capable of displaying a resolution of 1280×1024, then the configuration causes the refresh rate at 640×480 to be too high (in practice 90 Hz) for the i-glasses. The eventual solution was to configure X-Windows twice: once for a high resolution monitor and once for a low resolution monitor. The two configuration files were then hand merged, in a manner akin to a black art, so the single X-Windows session could drive both a high resolution monitor and the i-glasses. The settings, as represented by a line in the XF86Config file, for the VGA "standard" of 640×480 @ 60 Hz was determined to be:

```
Modeline "640x480" 25.18 640 664 760 800 480 491 493 525
```

The problem is that the configuring tools allow you to supply only two parameters: your monitor and graphics card. They were never designed to let you choose a arbitrary graphics mode of desired designation.

Observed Behaviour of i-glasses

A few experiments were performed to determine the response of the i-glasses for 640×480 graphics modes. The results are shown in Tables 8.1 and 8.2. Although the i-glasses can go up to 70 Hz, standard VGA was used because flicker is not so much of a problem with LCD technology, and because (presumably) the glasses are designed for this.

Vertical Frequency	Horizontal Frequency	Result
70 Hz	31.5 kHz	works
66 Hz	35.1 kHz	fails
61 Hz	32.5 kHz	works
59 Hz	31.5 kHz	works

Table 8.2: Viable scan frequencies for i-glasses

Drawing Methods

Given that the screen can be driven at the appropriate resolution, how can a stereo image be rendered? The list of possible approaches follows:

1. Draw to a window using X-Windows or some higher level library. The odd/even line ordering is critical to the headset display and windows that can be moved will destroy this. However, window positions can be locked.

2. Draw to the parent window using X-Windows or some higher level library. This is the simpler approach. The parent window is simply the full screen. Drawing to the parent window is performed in exactly the same way as drawing to a standard window under X.

3. Use a graphics library that writes to the raw screen. *e.g.* vgalib

4. Memory-map/Use the screen memory and write directly to it. This will probably achieve the highest performance.

5. Drive the screen using BIOS calls. May suffer from performance problems, as BIOS calls are driven off interrupts.

Clearly, this is a large and complex issue as there is a requirement to coordinate high-level rendering and low-level scanline rasterisation, and therefore it is dealt with separately in Section 8.5. None of the above approaches can be eliminated until a feasible solution is determined.

Scrolling

The final solution always places the image for the i-glasses display on the top-left corner of the X-Window virtual screen, so for satisfactory stereo operation the graphic cursor first may have to be pushed to the extreme left and extreme top of the display to bring the correct portion into view. An

attempt was made to knock-out this scrolling and to nail the hardware window always to the top left of the virtual screen. It was observed that the scrolling of the virtual screen occurs without an X-Window manager in operation. The conclusion is that the scrolling is done by the X-server, and indeed the precise position in the code where this scrolling occurs was located. This scrolling was disabled via a code modification. This achieved the desired effect. However, having to run a modified X-server on every machine on which the effect is required is rather onerous — given that the appropriate X-server source for each machine would have to be located, customised and then compiled. As a result, the standard X-server was simply used subsequently and in practice little problem was caused by having to wiggle the mouse to ensure proper stereo operation. Indeed, during pure test runs when no development work was undertaken, X-Windows was made to work in a native resolution of 640×480 (VGA mode). Here, there were no restrictions or requirements whatsoever on the positioning of the graphics cursor.

8.3.3 Graphics Hardware

At its most abstract, rendering the model of a Roman temple (say) turns a geometric description into a 2D raster image, reflecting the location of the current viewpoint. Ideally, to render the architectural models in a time- and space-efficient manner, graphics hardware should be able to render higher order graphical primitives such as Non-Uniform Rational B-Splines (NURBS [76]). Curved structures such as columns find their natural form of expression as higher order primitives, instead of being reduced to a mesh of polygons. PC graphics cards which support NURBS are only just emerging, but missed the window of opportunity when graphics hardware was being purchased, even though this occurred fairly late in the project. OpenGL does not currently directly support NURBS, so the software to access this hardware is an unresolved issue. However, NURBS are supported by GLU, which is a utility library for OpenGL. Currently, curved surfaces are rendered by GLU by being turned into a polygonal approximation to a specified level of detail. This is done in totally the wrong place because:

- the rendering hardware knows about pixel size, and should chose the appropriate level of detail itself.
- a NURBS is a compact representation, and it is clear that sending this to the graphics card rather than a mountain of polygons is faster.

Matrox G400	Voodoo 3
fewer limitations (*e.g.* texture size) good Linux support	slightly faster better Linux support

Table 8.3: 3D graphic card comparison

In the absence of such hardware, the choice of inexpensive state-of-the-art 3D graphics card was between the Matrox G400 and the Voodoo 3. The pros and cons are shown in Table 8.3. Eventually, the Matrox was chosen as the board had already been used successfully in-house. GLX (X with OpenGL extensions) supports hardware rendering into memory buffers. Thus it is possible to obtain the benefits of hardware acceleration whilst running X-Windows. The use of the Matrox G400 was found to considerably improve graphics performance over and above the original Rage 3D card in the development machine.

8.3.4 Input Sensors

There are orientation sensors built into body of the VR headset. Pitch and roll angles are measured by mercury level sensors that are appropriately oriented. Yaw is measured by a Hall Effect sensor that detects the Earth's magnetic field. The sensor information is relayed to a computer via an RS232 interface. The first software to support sensor input from the i-glasses under Linux appeared during the project at the following URL:

`http://www.weirdoz.org/iglasses/`

However, this Java software incorrectly reports that no head-tracker is connected. Further investigations, revealed that the test RS232 program under Java found no serial ports. Considerable effort was committed to locate the problem in the RS232 drivers, without success. To circumvent the problem, C-drivers for the headset sensors were written based on the Java source in a fraction of the time spent unsuccessfully debugging the Java RS232 drivers. The other circumvention route would have been to re-write the Java RS232 drivers.

8.4 Visualisation Software

A typical way of visualising a VRML model is to view it within a Web browser. A VRML "plug-in" for the browser provides the software mechanism for automatic visualisation. For Windows browsers,

a fairly standard VRML plug-in is CosmoPlayer and this was used to visualise the Roman buildings. CosmoPlayer was initially free, but was then charged for, so the CosmoPlayer installation was ultimately not upgraded. Since being abandoned CosmoPlayer has become free again. On Linux, no VRML browser plug-in could be located, so a public domain open source VRML browser was sought. The obvious candidate was FreeWRL as this provided the best support of VRML. FreeWRL is written in a combination of Perl and 'C' and talks to the Mesa implementation of OpenGL on Linux. However, FreeWRL was eventually discovered not to be up to the task, and the following sections details the work done with FreeWRL and the alternative candidates, until a suitable approach was encountered.

8.4.1 FreeWRL Development

FreeWRL is not usable "out of the box" for AR visualisation. Firstly the viewport is controllable only by the mouse or keyboard **not** by a position sensing technology. Secondly, FreeWRL only strictly supports VRML2, while our models were in VRML1. Given that no position sensing technology was available at the start of visualisation work, it was decided to use a joystick to emulate a position sensing technology. Attempting to integrate this emulated device for viewport control tests if FreeWRL is capable of supporting AR. This section describes the work undertaken to adapt FreeWRL for AR use.

Joystick Integration

The strategy for incorporating a joystick has two aims. Firstly it will test whether it is possible to take information from a (moderately) novel peripheral through to the internals of the code. Secondly, by translating joystick readings initially into viewport information and only then passing them to the core of the FreeWRL rendering code, a useful abstraction will have been achieved. The specially created software interface to achieve this separation of concerns is called the Virtual Position Interface (or VPI). Any device which supports this interface is called an Advanced Position Device (or APD). The initial thought was to code the VPI in VRML2 (using a Java Node) instead of having the code residing unportably in the browser. The flow of control would then become as in Figure 8.1 instead of the more traditional approach shown in Figure 8.2. Note that in FreeWRL, the 'C' code forms the low-level software supporting the high-level Perl code. However Java nodes under FreeWRL were found **not** to work (although JavaScript nodes did). As as alternative, VRware (a browser written in Java with public domain source) was downloaded and tested, as one might expect Java nodes to work

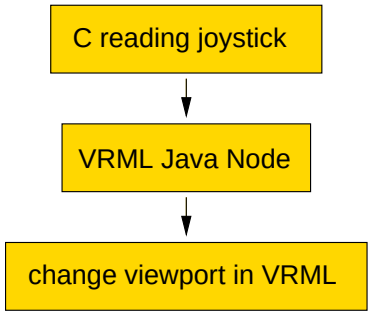

Figure 8.1: VRML Java node architecture

here. It was found to be slower than FreeWRL, though it has marginally better rendering quality and more control options via the GUI. Ironically however, VRware does not yet support JavaScript nodes and the Java nodes did not work! In the absence of a browser that supported Java, the less elegant traditional architecture was implemented, and joystick control was successfully accomplished through modifications to 'C' and Perl software.

VRML1 to VRML2 Conversion

FreeWRL only strictly supports VRML2, while the models were in VRML1. At the stage VRML1 to VRML2 conversion software was required, no public domain packages were available (this has now changed), so a manual conversion from VRML1 to VRML2 was undertaken. Fortunately, the Tcl source to produce the models is more tractable than the models themselves, and many of the models use a common set of Tcl subroutines, so the port was less onerous than it would have otherwise been.

Though VRML2 has greater functionality than VRML1, the changes necessary were actually at

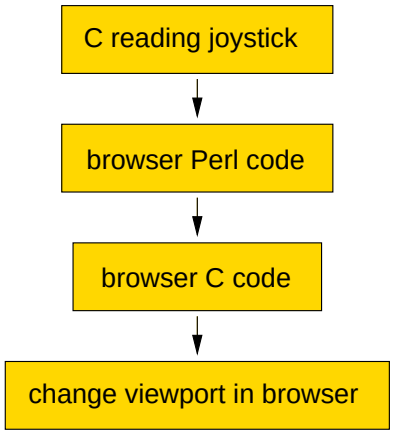

Figure 8.2: Traditional architecture

the level of graphics data structures. VRML2 uses a totally different structure hierarchy from VRML1 to hold the same data. The Tcl code changes were tricky, because the call structure of the original Tcl code reflected the format of these structures in VRML1. Essentially, unless the code was to be written from scratch, the VRML2 output had to be grafted onto an underlying software structure that was not appropriate. The eventual solution was messy, and involved the unfortunate use of state variables. These changes in data structuring within VRML appear to offer no particular advantage, and the negative impact through sheer incompatibility on the development of VRML software on many fronts was very clear. Such changes which produce severe incompatibility problems, should have been prevented at the outset. This is not to make a case against extensions to VRML, only against making changes outside a agreed extension mechanism. However, the experience gained of the Tcl code and VRML did prove unexpectedly useful later.

FreeWRL Limitations

Despite professing to support VRML Java Nodes, none of the test VRML models which use Java worked under FreeWRL. Various versions of Java and FreeWRL, were tried with no success, and it appeared that the qualitative critical advantage of VRML2 (*i.e.* programmable interactivity) was not going to be exploitable under Linux.

Furthermore, once FreeWRL was used on more complex models, such as a model of Essex University, visualisation bugs were revealed. For example, the display of non-convex polygons was incorrect — making the scene totally useless. In other models, strange effects appeared at the edges of polygons. The bugs encountered were documented and sent to the author, Tuomas Lukka, in the hope that future releases of FreeWRL would be suitable for our needs. These basic problems in visualisation was why FreeWRL was abandoned. However, FreeWRL was also not particularly fast on a 266 MHz PC. The model of the Gosbecks Temple complex was unusable as it took around 4 seconds a frame! Interaction breaks down at this slow update rate. Also it proved necessary to hold back from using some needed features of VRML2 as FreeWRL was not yet a complete implementation.

FreeWRL Support

There was always some concern over FreeWRL because, despite the undoubted competence of the work, it was the product of a single student and the lack of future support was always a possibility. In fact, some time after we abandoned FreeWRL, it did indeed go somewhat into the doldrums, as Tuomas was unable to support the product. Since then, it has been picked up by John Stuart at the Communications Research Centre Canada, and support and development is once again active. However, FreeWRL missed its window of opportunity for our system. Nowadays, the most competent VRML browser for Linux appears to be `VRMLview`, and this would be a better starting point for development if only it were open source! In short, although the VRML browser situation for Linux has improved, there is still not a suitable springboard for AR work.

8.4.2 CosmoPlayer Development

When the time came to couple the output of the VPS with a visualisation system the only viable VRML viewer was CosmoPlayer on Windows, while the VPS was running on Linux. However, a shared virtual world application written in Java had been developed at Essex University which worked using broadcast technology on the Internet for efficiency [77]. The visualisation engine was

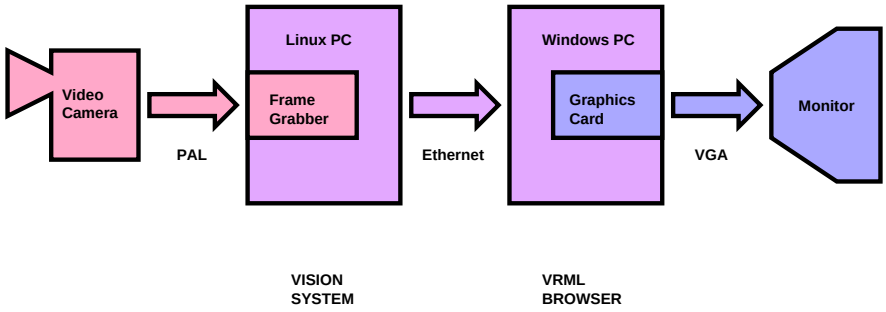

Figure 8.3: Split platform VRML visualisation hardware

CosmoPlayer running within Netscape Navigator. Java code is portable, so in theory it was possible to run the Java code simulating an avatar on a Linux workstation. The position updates for the avatar are broadcast on the Internet, thus updating the graphic display on the Windows platform. So by splitting the application across two platforms on the Internet, there is a route through to VRML visualisation. The scheme is shown in Figure 8.3.

Test Implementation

The source of the Java application was obtained, and a simple application deterministically moving an avatar was written to test the concept. The interfaces were undocumented so guesses had to be made as to how to achieve this functionality. This code did not work and much time was spent trying to work out the bug in using the supplied methods and classes. Eventually, the original author of the code was brought in to debug the system, and the problem turned out to be one of incompatible versions of the code (which would have been impossible to find even by groping into the innards of the Java). Once the version of the source used was regressed, the small test application worked immediately to

much relief! It is observed that most software problems can be quickly resolved by having the original author to hand, and this is a non-trivial argument for in-house rather than centralised development.

Pipelined Implementation

The VPS was connected into this network solution, using a simple-minded UNIX pipe to take the output of the VPS as input to the small Java application. The hybrid system worked, but the latency of this first implementation was unacceptable. A movement of the camera used by the VPS, would be followed 30 seconds later by a movement of the avatar on screen. What was going wrong? Position events were undoubtedly being blocked in the UNIX pipe. On the face of it, the CosmoPlayer browser was more than capable of rendering the four frames a second that was necessary to support the four position fixes per second of the VPS, so there seemed to be no valid reason for such a backlog.

Single Process Implementation

To circumvent the latency of the UNIX pipe, a second single process implementation was undertaken using a 'C'-to-Java translation layer to communicate positional data. Only sketchy and incomplete documentation was located on how to effect inter-language operation in this direction. It is not a trivial operation as the Java Virtual Machine has to be explicitly started, even before inter-language calls can begin to make sense. Trial and effort was involved, and the process was documented to save others the same problems in future (see Appendix M). In contrast there is much documentation on how to call 'C' from Java, because it is correctly assumed that the more frequent occurrence will be to invoke a so-called "native method" from Java to either gain a performance improvement or to make some system call — 'C' being the *lingua franca* of system calls on most platforms. Latency was reduced to around 5 seconds — not acceptable for viable AR but at least the result was demonstrable.

Angular Orientation

The system was extended to use angular information from the VPS additionally to change the avatar's orientation. The avatars available were non-articulated, so a human attached to the VPS looking up would cause the avatar's legs to rise off the ground in an unnatural fashion, so the system was restricted to using the yaw angle! In fact, satisfactory functioning using the pitch and roll angles was not achieved. Some attempt was made to ensure that the orientation of the VPS camera correctly controlled the orientation of the avatar, but mapping between the orientation matrix generated by the

VPS and the orientation notation of VRML (based on an orientation axis, and a single rotation angle) was a difficult problem. For a start no ambiguity-free definition of the VRML orientation conventions could be found and certainly not one that involved some helpful maths. Debugging the system at this stage would have been painstakingly difficult as the two machines involved were physically separated, and a debugging cycle involved a lot of leg work: changing the camera orientation at one machine then observing the resultant change in avatar position at the other. The orientation problem was stacked as a "matter pending" due to all these difficulties, and because the only useful angle (yaw) actually worked perfectly! Note that later the orientation problems were resolved under more suitable circumstances. Indeed this turned out to be one of the most difficult problems tackled, and its resolution would have been nigh on impossible in the CosmoPlayer environment.

Hybrid Architecture

It was worked out that if any of the software components on the Windows PC was upgraded, be it Windows, Netscape, Java or CosmoPlayer, then the software would no longer work. Indeed, the CosmoPlayer plug-in can no longer be loaded off the CosmoPlayer site. In fact, that particular Windows machine had been kept in aspic so the shared virtual world demonstrator would continue to function. A central coordination process called the "GateKeeper" deals with the sharing aspects of the virtual world. This is written in Java, and it was eventually moved from the Windows platform to a Linux one, in an attempt to reduce system dependencies. However, the visualisation side of the hybrid architecture remained as fragile as ever. The software architecture is shown in Figure 8.4. The next stage was to enhance the system to use information from the VPS to change the viewport position, rather than simply move an avatar. However, the following factors began to weigh rather heavily:

1. The effort in debugging such a system.

2. The hybrid nature of the system. The AR demo is exceptionally vulnerable as it depends on many system software components: 'C' under Linux; 'C' library for interconnecting with Java under Linux; Java under Linux; Java under Windows; Netscape under Windows; and CosmoPlayer plug-in under Windows.

3. The ultimate limit on performance of such an architecture. Reducing the 5 second latency would have been either challenging or impossible.

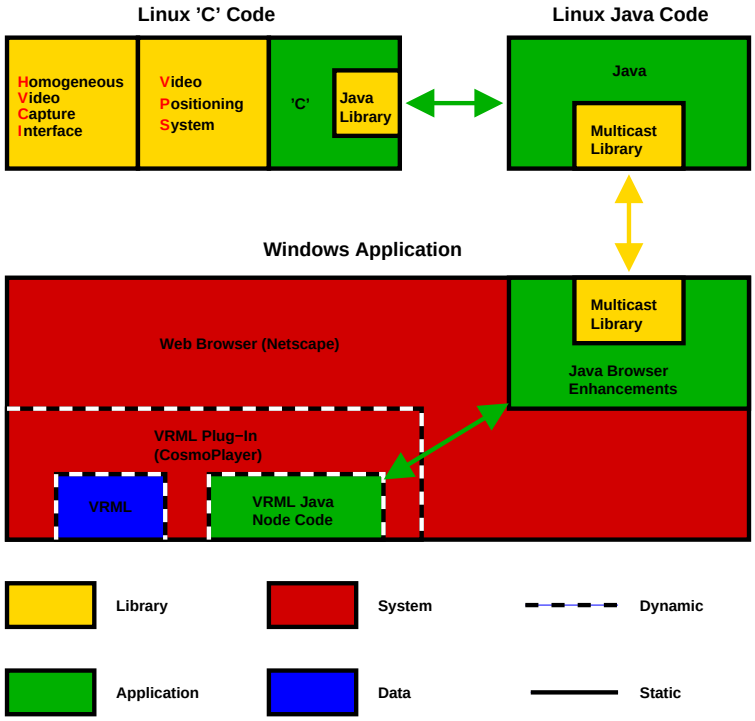

Figure 8.4: Split platform VRML visualisation software architecture

4. The seeming impossibility of resolving the orientation problem. Much effort had been made, before losses were cut.

5. It was unknown how to alter the viewport. Clearly, the shared virtual world demonstrator would have to be extended to support this, and the mechanism to dynamically change the viewport within VRML using external data, is by no means trivial. This problem was researched and no immediately clear answer was found. Solving this problem most effectively (in the absence of a fully functional Linux VRML browser) would involve development work on a Windows PC.

6. Lack of support for the i-glasses. CosmoPlayer does not support stereo graphical output for the VR headset in use.

Conclusion

The interoperation of different pieces of software across the Internet is often presented as easy, and VRML in particular is also claimed to be a doddle for producing network-transparent interactive 3D

applications. In theory this is great, but in practice many unavoidable problem were thrown up by the nature of AR development. This is why the use of CosmoPlayer in a hybrid architecture was abandoned. Taking a higher level view, these difficulties fall into two categories:

1. The "Tower of Babel" problem. So many languages and/or system dependencies are involved, yet the upgrade of a single one of these (or even a single bug in one of these components) will cause the whole structure to collapse. Even if the component technologies are bug-free, often the sheer difficulty in interfacing the systems can lead to defeat — even though the application and underlying mathematics are themselves totally understood.

2. Novel Peripherals. Trying to do anything new (such as adding qualitatively new types of peripherals) or anything unconventional (changing the 3D viewport by program control outside of a GUI) is just not possible using existing systems. To achieve this, one has to go into the guts of other people's software, such as with FreeWRL and the shared world demonstrator. An abstraction such as VPI would help in these circumstances, and any 3D browser should be able to be driven (viewport or object position) by any positional device. This is currently **not** possible, and most applications are hardwired in mindset and in practice to keyboard and mouse. Even a joystick which provides navigational intuition through its form, may be used in games but **not** in 3D browsers.

8.4.3 OpenGL Development

Why OpenGL?

Before the CosmoPlayer path was followed beyond the point of no return, it was realised that to complete the visualisation software within tractable time, and for it to be practical, it would have to be written from scratch and outwith a VRML framework. The level typically below VRML is OpenGL, a graphics API, based on Silicon Graphics's proprietary GL. OpenGL/GL is an exceptionally good portability layer as it works across platforms. It is also good in performance terms as hardware is optimised specifically to support OpenGL. Despite its age, OpenGL is the *de facto* interface supported by the latest generations of fast 3D graphics cards for PCs. The move from VRML to OpenGL as the representational form is quantitatively from a higher to a lower level and qualitatively from a description of a potentially interactive scene to imperative code to draw this scene.

Software Architecture

Two items of software need to be constructed to support this architecture:

1. OpenGL code to draw a Roman Temple (say);

2. a framework which can accept viewpoint information in real-time and the OpenGL code above, to produce a dynamic rendering of the Roman Temple from the supplied viewport.

How can the VRML model be turned into an OpenGL version? The immediately obvious assumption is that a VRML parser would be required. No public domain VRML parser existed at this stage. Significantly, just such a VRML parser has subsequently been made available by the University of Washington, having been developed out of their AR work. This development is confirmation of the unsuitability of existing VRML tools for AR. Indeed, at many stages developing a VRML parser was seriously considered. However, although this would have been of much wider usefulness in the community, much work would have been involved. It was also realised that there is a much simpler solution to achieve the same end. The VRML model for Gosbecks is around 6000 lines long, whereas the Tcl to produce this VRML consists of 1000 lines specific to the Gosbecks model and 2000 lines of generic support code. Clearly, converting the Tcl to produce 'C' code calling OpenGL would be more efficient in the longer and shorter terms than converting the VRML directly.

Conclusions

The Tcl was successfully converted to generate OpenGL as well as VRML. This is represented diagrammatically in Figure 8.5. A few observations can be drawn from the experience of conversion:

1. Abstraction Level. VRML and OpenGL are surprisingly not significantly different in levels of expression, and there are in most cases one-to-one correspondences between the constructs in the languages, where non-interactive visualisation is concerned.

2. Conversion. Automated conversion between two languages, requires knowledge almost at a formal level for both languages. It was much harder to obtain such information for VRML — and ambiguity resided in many VRML constructs, in particular the representation for orientation. Fortunately, some small previous experience had been obtained of both VRML and OpenGL, but even then much had to be established by trial and error.

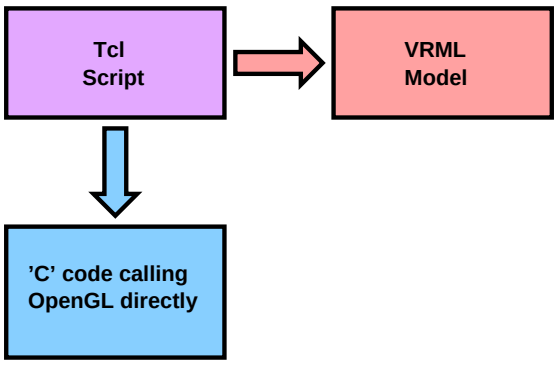

Figure 8.5: Automated production of OpenGL and VRML

3. What was Missing? Some additional OpenGL information had to be supplied (most particularly lighting) in order for the OpenGL model to be visible. Such presentation-based information is normally specified within the GUI interface for a VRML browser. However, developing an OpenGL browser of equivalent functionality would have been too great a software undertaking. Instead, very simple-minded lighting was hardwired in the code to ensure that the OpenGL model would be visible.

4. Obtaining Output. The Tcl was altered to produce 'C' code calling OpenGL, in addition to (rather than instead of) the VRML code. It is advantageous to stick with a single version of the source code, particularly as it operates in a batch manner with little regard for performance. All lines of OpenGL code were prepending by the "OPENGL" keyword. Postprocessing tools disentangle the output into VRML code and into OpenGL with this keyword extracted.

5. File Formats. How is the AR system going to load an arbitrary OpenGL model? The problem is that there is no file format that goes hand-in-hand with OpenGL. There are no "load scene

from file" or "save scene to file" calls within OpenGL. In contrast the VRML *is* the storage file. Initially to circumvent this problem, the OpenGL code automatically produced, was compiled and linked (essentially hardwired) into the application. Calling an `init_model` function, within this 'C' code, sets-up an OpenGL segment, and places the code which draws the scene inside this segment. The identifier (an integer) of the segment is the return value of the function call. Anytime the AR application wants to redraw the scene, it calls `draw segment` with the appropriate identifier as argument. Eventually, it was realised that there is actually no need to define a file format, though initial attempts were made at a design. Provided 'C' code for a model can be dynamically loaded by the application, then the effect is identical to the dynamic loading of a data file. This is more generally a mechanism for creating a portable file format for data structures set up by an API, where there is **no** official file format. The 'C' code could even be compiled on the fly before the loading of the corresponding object file: further extending the similarity of the approaches.

It is interesting to observe that the resulting 'C' code to draw Gosbecks is around 10,000 lines. It is a great comfort that this was converted at a meta-level rather than laboriously line-by-line. The 'C' code has had its loops unrolled (following the VRML approach), and so is far less compact than it could have been. On the other hand, if the Tcl code (rather than its output) had been converted to 'C', then the more compact expression would have been retained. In the last analysis the loops are unrolled only on initiation while setting up the internal OpenGL data structures so any effect on performance is not an issue.

Ultimately, the ideal interface to graphics hardware should permit some notion of iteration for compactness of expression, and ultimately move towards a more general form of algorithmic encoding. An example of this is NeWS window system where all graphics is performed by PostScript programs. PostScript is a fully-featured programming language as well as a storage format for graphics, so it is capable of being used in precisely this manner. To draw 100 Roman columns with OpenGL, the drawing process must be repeated 100 times. However, to try to obtain some benefits from iteration, all objects that are drawn more than once are placed within their own graphics segment, so only a segment drawing call is necessary each time instead of repeating the component operations.

8.4.4 Togl Development

To minimise the software effort involved in developing an OpenGL browser that accepts dynamic viewport information, a decision was taken to use Togl. Togl is a toolkit that allows the rapid development of OpenGL programs controlled by a Tcl GUI. Togl applications are written in a mixture of 'C' code calling OpenGL and Tcl/Tk. In particular Togl handles the mechanism for inter-language working and supplies a straightforward context (a Tcl panel) within which OpenGL graphics are presented. There is no magic to Togl. It comes as a 'C' source file that is linked into the application. The application writer constructs Tcl code that sets-up widgets and glues these together with the 'C' code. Sample applications are supplied as a starting point. Version 1.4 of Togl was downloaded — this crashed at run-time. Version 1.5 of Togl was then downloaded. The compilation failed, but manually guessing at "suitable" code for the point of failure in the source enabled Togl to compile and successfully operate.

An initial OpenGL browser was developed under Togl. As the simplest mechanism available, sliders were first used to control the position of the viewpoint but it soon became obvious that these were far from intuitive. It was considered worthwhile developing a practical navigation mechanism using mouse input, because it would provide a solid foundation on which to place input from genuine position measuring devices. Also, a mouse can still profitably be used manually to align the virtual and real worlds, even while a position device is in use.

For intuitive navigation, it was decided to copy an existing practical approach rather than invent one, so the navigation interface to CosmoPlayer was studied in order to clone or reverse engineer it. While experimenting with the interface to discover exactly what it does, it was noticed that the interface does fail on occasions and some of the clearly intended mappings of the mouse to viewpoint location fall apart. It proved considerably harder than anticipated to get working navigation code, and in this light the mistake in CosmoPlayer was much more understandable.

A basic implementation issue is whether to keep viewport position states in the form of matrices or in the form of positions and viewing angles. The matrix form is more convenient, as an incremental change in the viewpoint just requires the multiplication of an accumulator viewport matrix with the incremental matrix. In addition, the OpenGL API uses a matrix representation. However, over time it was felt that after the accumulation of a large number of incremental multiplies the position matrix may through floating-point inaccuracies lose its orthogonality and become ill-conditioned. The end result would be a subtly distorted scene. On the other hand, maintaining position and viewing angle

would allow the position matrix to be generated anew each time keeping it “fresh”. In the end the solution developed was somewhat hybrid despite trying to avoid the use of matrices until the last possible moment. In fact, due to the difficulties involved, the first working solution obtained was not taken any further. The matrix representation used means that tired matrices could result, though no problems were experienced in practice. Ultimately the solution should be taken to a full matrix-free form.

The execution speed of Togl was good, as graphic-card specific (*i.e.* Matrox Millennium G400) OpenGL drivers were being used and no major performance hit was encountered as a result of Tcl being an interpreted language.

8.4.5 GLUT Development

As the complexity of the OpenGL browser code began to increase, Togl started to become a millstone rather than a benefit. Any change to the GUI required corresponding changes in both ‘C’ and Tcl codes, which complicated matters and severely hampered development. It also became clear that the application was going to require more than one OpenGL window, at least during development for debugging and other purposes *e.g.* one window for the right eye viewport and another window for the left eye viewport. Togl cannot support more than one OpenGL window. Ultimately, of course, the images would have to be combined in a way suitable for the 3D display technology being deployed. What software effort was saved for a simple application through the use of Togl was now being lost for an increasingly complex application.

Due to the bugs and limitations encountered, the decision was taken to move the software development entirely to ‘C’ using the GLUT interface to provide the context for OpenGL rendering. GLUT is the *de facto* standard library supplying context for OpenGL rendering, and it is provided along with OpenGL. It is widely available on a variety of different platforms. Although the port from Togl to GLUT required considerable software investment, subsequent development was able to continue at a much increased pace.

8.4.6 GLX Development

Eventually, the time came to merge the right and left eye images. Line-by-line interleaving is accomplished in OpenGL by using `glReadPixels` repeatedly to read a pixel row from the right and left eye images in turn and then to use `glDrawPixels` to write each of these contributed rows to

the target image. The right and left windows in which the right and left images are drawn should be "unmapped" in the parlance of the X-Windowing system *i.e.* they should be taken off the screen and simply regarded as intermediate buffers for drawing that are not intended to be seen. Unfortunately the GLUT interface was not capable of removing these images from the screen, and attempting to draw into graphic buffers instead of windows simply did not work. Furthermore, `glReadPixels` and `glDrawPixels` were excruciatingly slow, and all possibility of operation at interactive speeds was lost using these functions.

The RasterOp operations of OpenGL are too slow. The basic RasterOp operation of OpenGL is exceptionally complicated, allowing very many different possibilities of translating pixels from one representation to another in the process. All that is required for row-merging is a simple copy — but all OpenGL provides is an operation that is total overkill. The geometric primitives of OpenGL are elegant and sufficient. On the other hand the pixel model for RasterOp operations is clumsy and heavyweight, reflecting the nature of graphics hardware at the time GL was developed. This hardware model consists of a finite number of off-screen image buffers, a dedicated accumulator buffer and a dedicated depth buffer, *etc.* . This hardware model is largely redundant.

The Mesa OpenGL code for `glReadPixels` and `glDrawPixels` was studied, and many inefficiencies were identified. With relatively little effort both operations were speeded-up by a factor of two, as there was much redundant copying of image data back and forwards through the layers of software. Even so, the performance was nowhere near satisfactory and an alternative direction was sought. Mesa OpenGL is a considerably less mature product than GL, and it would be true to say that unless the code is on the Quake critical path, it was very far from optimal. Quake is a 3D graphics intensive "shoot-em-up" game which has been largely responsible for driving the development of powerful 3D graphics hardware for PC-gaming. More recently, the efficiency of the OpenGL implementation has been considerably improved but it is doubtful, given the complex specification of `glReadPixels` and `glDrawPixels`, that it could perform adequately even given the current levels of hardware.

The software layer underlying GLUT on most X-Windows-based machines is called GLX. This layer provides a context on an X-Windows system for OpenGL rendering. The reason GLX was initially avoided is because it is X-specific so does not provide the portability of GLUT. It is also considerably more complex than GLUT and does not possess the wealth of tutorial material of GLUT. However, in order to obtain access to the faster RasterOp operations of X, the implementation was moved to GLX. This permits access to the X-Window ID of each OpenGL window. Using this X-

Window ID, native X-Windows operations can be performed to yield higher performance.

Moving the software from GLUT to GLX was a rather daunting software task, so to ease the transition a re-implementation of GLUT was written on top of GLX, that placed the essential information (such as window-IDs) into global data tables. These global data tables allowed the application to access the X-information required without greatly disturbing the existing use of the GLUT interface. Only a few GLUT calls were extended by a few extra parameters.

One may ask why the GLUT was re-implemented when the source of GLUT based upon the GLX was available? The answer is that the full GLUT implementation is large and confusing to work with, and it would be difficult to acquire the X-information required — some of the information was available through the API but other parts would have to be teased out via software modification anyhow. Eventually, a GLUT implementation in GLX was written that provides all the functionality required by the application. The performance of the X-Windows operations `XGetImage` and `XPutImage` was compared informally against `glReadPixels` and `glDrawPixels`. Despite the improved performance, the application still ran below interactive rates, and another approach was required.

8.4.7 Hardware Driver Development

Whatever the problems encountered enticing system software to perform efficient image interleaving. the optimal way to merge the left and right eye images is clear. Assume the required stereo image is $w \times h$ pixels. Draw the right and left images anamorphically reduced vertically by a factor of 0.5 horizontally adjacent to one another in a buffer of $2w \times \frac{h}{2}$ pixels Then using a simple RasterOp (with a source scan line increment of $2w$ and a destination scan line increment of w) the interleaving can be achieved in a single efficient operation as in Figure 8.6. Since double buffering is being used, a RasterOp is necessary to bring the rendered image to the screen anyway, and it makes sense for this refresh RasterOp to be the one described. In short, no extra overhead need be introduced to produce a specific scanline-interleaved image. In fact, with this strategy the overhead of a stereo image is less than the overhead of producing and displaying two images separately. In the stereo case the images are half size, reducing the rendering overhead and as explained only one refresh RasterOp is required instead of two. Indeed, with hardware double buffering even the RasterOp is not necessary, and the same memory is just regarded as having a different scanwidth for OpenGL rendering and hardware display.

Unfortunately, OpenGL does **not** permit differing source and destination scanline increments.

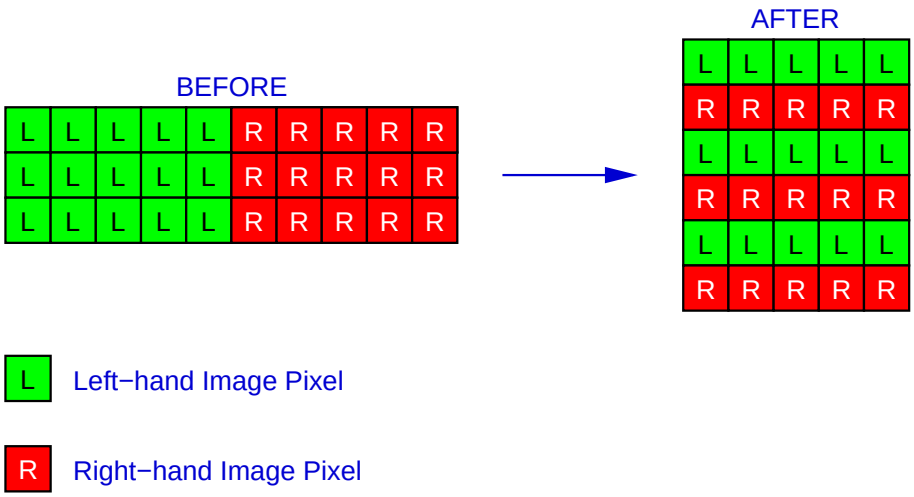

Figure 8.6: Interleaving algorithm

This is possible using X-calls but this did **not** work in the GLX environment, and anyhow to gain access to this refresh RasterOp entails entering the X-driver code, because it is a system operation rather than an application one. So after much effort to avoid entering graphic-card-dependent code, the decision was finally taken to modify the drivers at the very lowest level possible *i.e.* the level of hardware registers on the graphics board. In fact, considerably less time was taken to get this working than had been spent trying to avoid taking this step, and again with hindsight the resulting performance gain for comparatively little software effort more than justified the endeavour.

Debugging this code was exceptionally difficult because a graphical debugger requires this X-code being debugged to operate! Thus the code had to be debugged across two machines, the remote machine running the debugger and the local machine running the (modified) low-level X-driver code. This set-up considerably sped-up development and revealed that the initial understanding of the driver code has been correct, but due to difficulty in picking up the right version of the driver (this hinges on environment variables) wrong conclusions has subsequently been drawn about the code on a single machine testing platform. Here code changes could catastrophically destroy the windowing system

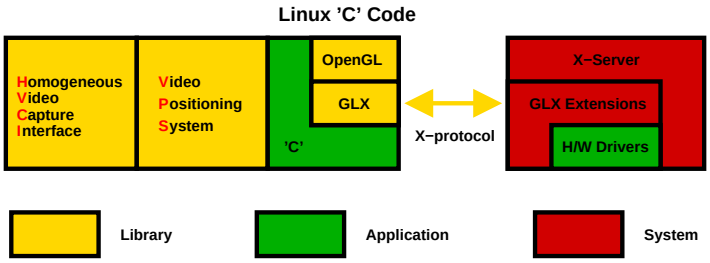

Figure 8.7: Final software architecture based on OpenGL visualisation

and/or operating system. The drivers are very flaky and the whole machine would often crash (quite irreproducably) for no apparent reason, causing an infuriating 15 minute re-boot cycle.

With such a modification the Temple dedicated to Claudius model runs at interactive rates in stereo. The Gosbecks model runs at around 2 frames per second. This equates to around 3.9 frames per second in mono. The fastest the Gosbecks model had run previously was 1 fps on a 3D graphics card under Windows running CosmoPlayer. The performance can be considered state-of-the-art for such a geometrically complex model. Though the Roman models do not appear to be drawn as quickly as, say, a scene in a computer game such as Quake, it has to be remembered that the Roman models are exceptionally complex geometrically, whereas the apparent complexity of a Quake scene is actually all due to textures. Though the rendering software is being kept in its current form, future work could involve undetectably reducing the geometric complexity and adding texture to improve performance and the "perceived" image quality.

Having modified the Mesa driver complicates code updates on the machine, because all subsequent source has to be similarly modified. This is the drawback of working underneath a well-defined

Task	Operation Time	
	Simple Figure (150 lines)	Complex Figure (2400 lines)
draw mono image to window	128 ms	139 ms
draw stereo image to buffer	192 ms	294 ms
copy buffer to window	2000 ms	2000 ms

Table 8.4: Mono *vs.* basic stereo performance

API. Basically, this version of the hardware driver for GLX is only suitable for rendering stereo images. It would be bad software engineering practice, though feasible, to put special flag values in certain system calls to switch between stereo and mono operation. The final software architecture is illustrated in Figure 8.7, highlighting the critical changes made within the GLX extensions to the X server.

8.5 Stereo Rendering Performance

The critical performance issue is interleaving the left and right eye images as this is outwith the usual gamut of OpenGL operations. Achieving the "filigree" effect of the appropriately interlaced left-eye and right-eye images could easily overly impact on graphics performance if care is not taken. This section reports on the quantitative performance evaluation of the different approaches, using a 640×480 image, profiling separately the two stages in stereo rendering:

1. drawing right and left eye images into a buffer

2. interlacing right and left eye images on the screen on a line-by-line basis

8.5.1 Stereo *vs.* Mono

The baseline performance of drawing a standard or monoscopic image to a window was measured against that of the most straightforward approach to stereo: using the OpenGL `glDrawPixels` operation for moving rows of pixels to the screen. Measurements were taken for both a simple and a complex scene. Optimised hardware drivers were not in use. The results are shown in Table 8.4. The figures show that there is a huge and unsupportable bottleneck in refreshing the image to a window using the `glDrawPixels` command in OpenGL. Clearly there is gross inefficiency because all the

Drawing Complex Figure (2400 lines)			
Task	Generic Driver		
	Window Buffer		User Buffer
	- DMA	+ DMA	+ DMA
	Approach 1	Approach 2	Approach 3
draw mono image to window	139 ms	38 ms	40 ms
draw stereo image to buffer	294 ms	78 ms	56 ms
display stereo image in window	2000 ms	417 ms	206 ms

Table 8.5: Window *vs.* buffer performance for generic OpenGL driver

graphics timings are double buffered, so all are already subject to a (much more efficient) memory copy to the display.

8.5.2 OpenGL Buffer *vs.* User Buffer Rendering

Is it more efficient to render into user-space instead of the more usual window back buffer? What about the use of DMA? The performance results for the three approaches below are presented in Table 8.5 for the generic OpenGL driver:

1. window memory rendering, interleaving with `glReadPixels` and `glDrawPixels`
2. window memory rendering using DMA, interleaving with `glReadPixels` and `glDrawPixels`
3. user memory rendering using DMA, interleaving with `glDrawPixels`

The use of DMA and a user-level buffer is beneficial, but overall stereo performance is still pretty slow. However, user buffering will have to be abandoned anyhow as hardware rendering into user memory is not supported by the dedicated Matrox G400 driver. As well as the G400 driver not being able to support approach 3, it also falls over with approaches 1 and 2, irrespective of the graphics display mode. Even the generic driver was found to render improperly in 24-bit mode. These failures are particularly ironic given the fact that everything is written in pure OpenGL. In short there is a serious bug in the G400 driver: no form of benchmarking is even possible and there was nowhere left to go.

Task	Generic Driver `glReadPixels`/`glDrawPixels`	Generic Driver `glCopyPixels`
draw mono image	38 ms	25 ms
draw stereo images	61 ms	67 ms
interleave stereo images	98 ms	68 ms

Table 8.6: Read/Draw pixels *vs.* copy pixels

8.5.3 OpenGL Buffer *vs.* User Buffer Interleaving

Using the generic Mesa implementation of OpenGL on Linux, interleaving in user address space was compared to interleaving in OpenGL drawing space.

A. User Space Procedure:

1. `glReadPixels` (single call) into user space
2. interleave in user space
3. `glDrawPixels` (single call) into OpenGL space

B. OpenGL Space Procedure:

1. Multiple calls to `glCopyPixels`

Approach B should be faster than approach A and this is indeed the case for the generic Mesa driver. However, it was impossible to carry out the timings for the Matrox Millennium G400 Mesa driver because it would always crash, bringing down the windowing system or the machine. Note that `glCopyPixels` can only work within a single OpenGL buffer not between buffers as was required. Attempts to copy between buffers fail. In consequence the code had to be altered greatly to enable the use of `glCopyPixels`. Results, given in Table 8.6, show that, as expected, copying is generally faster than reading then writing. Surprisingly, when the G400 driver was tried again on the off-chance, it did not fall over! Investigations proved this was because only one window was in existence, and the G400 driver gets confused when more than one OpenGL context is in use.

8.5.4 Generic *vs.* G400 Driver

Now that the catastrophic limitation in the G400 driver had been discovered, it was possible, with care, to benchmark this comparatively against the generic driver. Results are given in Table 8.7. The

Task	Generic Driver Copy Pixels	G400 Driver Copy Pixels
draw stereo images	41.8 ms	0.517 ms
interleave stereo images	49.8 ms	169.5 ms
swap double buffers	20.8 ms	0.0326 ms

Table 8.7: Generic *vs.* G400 drivers

OpenGL Operation	X Operation
`glReadPixels`	`XGetImage`
`glDrawPixels`	`XPutImage`
`glCopyPixels`	`XCopyArea`

Table 8.8: Equivalent OpenGL and X operations

G400 driver can draw and swap buffers hundreds of time faster than the generic drivers. However, interleaving using `glCopyPixels` is 4 times slower! This is a **huge** bottleneck, causing the fast driver to be slower overall. It is clear that while the commonly used operations may go lightning fast the less common operations are a disaster area *i.e.* the G400 driver's implementation of `glCopyPixels` is unusable, limiting the entire system to run no faster than 5 Hz.

8.5.5 OpenGL *vs.* X

A final attempt was made to circumvent the bottleneck by moving to the GLX library for establishing OpenGL contexts. Using GLX, the X identifiers for the windows being used are available, so it becomes possible to use X operations instead of the OpenGL ones identified as inefficient. The corresponding operations are shown in Table 8.8. Although formal benchmarks were not taken, it was clear that there was not going to be a significant improvement in speed that would do any form of justice to the rendering capability of the G400 board. The point of diminishing returns had been reached. A small edge in performance could be gained, but it was still going to fall far short of what was required. It became clear that a totally new approach was necessary. This is described in Section 8.4.7 and removes any necessity to benchmark because it is the optimal performance solution.

8.6 Drawables

The problems described above result from excessive variety in the types of the object that are actually drawn to. Certain operations may only apply to certain object types, and certain objects may simply be unsupported. The situation is indefensible give that the low-level representation within each object (*i.e.* a 2D pixmap) is actually the same across the object types. To help resolve the confusion, a study was made of these objects and some attempt was made to bring order to the chaos.

8.6.1 Introduction

When working with X, OpenGL and GLX (X + OpenGL), various forms of raster image emerge. The confusion has two origins:

1. CPUs and graphics processors are different. CPUs are not in general particularly efficient at performing graphics. Historically for basic rendering and nowadays for high-end rendering, specialised graphics processors are employed. Graphics are either done in a slow but fully flexible manner in the CPU's address space by user code or quickly in the graphics processor's address space by specialised code. To transfer data between the two address spaces generally has to be done explicitly. Unfortunately, even if the user **could** access the address space of the graphics processor, it is unlikely that the graphics API would reveal its implementation information such as the addresses at which the images are stored.

2. Networked graphics systems. With the graphics processor and CPU sitting on different machines, address spaces are perforce separate, and there are explicit commands to transfer image data.

8.6.2 The Problem

How can one achieve efficient integration of graphic processor and CPU drawing without incurring transfer operation penalties? The final display operation can also be regarded as a transfer operation. The client/server model leads to a combinatorial explosion of ways to perform a task — the most efficient combination of server-side or client-side operations can never be guaranteed. A transparent virtual memory solution, where it would be always possible to twiddle the bits manually if required, would be ideal — no matter where these bits are! The distinction should only be between whether

User Space Image Format	Transfer Operation	Graphics Space Image Format		Display-able	Render-ing	Internal Graphics Transfer
N/A	N/A	1	OSMesa::user buffer	NO	OpenGL	N/A
user buffer	`glDrawPixels`→ ←`glReadPixels`	2	OpenGL::system buffer	YES	OpenGL	`glCopyPixels`
XImage	`XPutImage`→ ←`XGetImage`	3	X::X Window	YES	X	`XCopyArea`
XImage	`XPutImage`→ ←`XGetImage`	4	X::XPixmap (>1 bits per pixel)	NO	X	`XCopyArea`
XImage	`XPutImage`→ ←`XGetImage`	5	X::XBitmap (1 bit per pixel)	NO	X	`XCopyArea`
user buffer XImage	`glDrawPixels`→ ←`glReadPixels` `XPutImage`→ ←`XGetImage`	6	GLX::GLXPixmap	NO	X OpenGL	`XCopyArea` `glCopyPixels`
user buffer	`glDrawPixels`→ ←`glReadPixels`	7	GLX::GLXPbuffer	NO	OpenGL	`glCopyPixels`

Table 8.9: Drawable object summary

an image is in a rasterised form or not. OpenGL tries (admirably) to abstract itself away from a representational form in the most part, and its remit does not cover any form of direct raster access.

8.6.3 The Drawables

The objects which can be drawn to are summarised in Table 8.9. The naming convention for these objects is <system name>::<name within system>. Notes relevant for interleaved stereo rendering are provided on these object types below:

1. OSMesa::user buffer

   One extension to OpenGL (the “OSMesa” calls) allows OpenGL to render directly to user memory. Unfortunately this extension is not supported by the Matrox Millennium graphics card driver, and was not particularly fast in the generic OpenGL implementation.

2. OpenGL::system buffer

   This is the standard place where OpenGL draws. The user is not able to access these buffers directly.

3. X::XWindow

   The standard place where X draws.

4. X::XPixmap

   X can also draw to memory on the X-server in the form of a Pixmap but **not** directly in X-client memory.

5. X::XBitmap

   An XPixmap which is one bit deep.

6. GLX::GLXPixmap

   The GLX version of an XPixmap. Both X and OpenGL can draw here.

7. GLX::GLXPbuffer

   Another off-screen server memory image where GLX can render (but this time not X). This is more flexible than a GLXPixmap, and does more of what is required for stereo rendering but it only in the specification of OpenGL 1.3 *i.e.* no implementation has been found.

8.6.4 OpenGL Context Conundrums

As well as the basic confusion over drawables, particular software incompatibilities between the generic and G400 drivers are based on the related issue of OpenGL contexts. OpenGL is not self-contained and does not cover the setting-up of "contexts" within which rendering can occur. Instead, reasonably portable methods of opening windows and switching between double buffers and the like, are available through the use of OpenGL toolkits such as GLUT. Looking below the GLUT interface, which open source makes possible, reveals that a number of different primitive low-level contexts are available within a generic Mesa implementation:

1. Fake_glX
2. OSMesa
3. Real_glX
4. SVGAMesa
5. XMesa
6. glX

The nature of these different contexts will not be described, but as an illustrative example the OSMesa context is the particular one used by the Mesa demos for rendering into an area of memory rather than the front or back window buffer. The OSMesa context could be used to minimise processing, as the rendered image is immediately accessible within user memory space (for interleaving) so does not first have to be grabbed using an inefficient `glReadPixels` operation. Despite the variety of low-level contexts available in the generic Mesa implementation, only one limited context is available in the Matrox Millennium G400 Mesa implementation, namely GLX. This removes various mechanisms to obtain good performance. The initial stereo rendering experiments had been made using the OSMesa context, and were not transferable to GLX. Hardware rendering of the G400 is used only when drawing to a back window buffer. As a back buffer is not user-accessible, subsequent operations are restricted to pure OpenGL. Basically, there is no route to high performance stereo rendering. The irony is that a modern computer has a flat unified address space so all the system software limitations encountered are completely unnecessary!

8.7 Visualisation Mathematics

This section describes the mathematics behind AR visualisation. Strictly speaking this should be no more than the mathematics of projection which have already been established. However, there are applied issues of how to reconcile projection within OpenGL with that required by the VPS. The reconciliation proved demanding, and even fed back to changes being made in the VPS so orientation semantics were well defined. A strategy for achieving this reconciliation (*i.e.* establishing alignment within an AR system) is presented in Appendix L, because it previously eluded some other researchers at the university. Also, a mathematical algorithm to control a viewport using a non-positional device is presented, both with and without a genuine positional device being present.

8.7.1 Intuitive 3D Viewport Control by Input Devices

The algorithms devised to provide intuitive 3D viewport control using input devices proved to be considerably more tricky than expected to develop. This was particularly the case where two input devices (*i.e.* mouse and headset orientation sensors) were allowed to update the viewport simultaneously. The difficulty is to get the devices operating together coherently.

MODE	Mouse Movement	Viewport Change	Variable
1	horizontal (+x) vertical (+y)	- yaw move in	d_{yaw} $z_{translate}$
2	horizontal (+x) vertical (+y)	move left move down	$x_{translate}$ $y_{translate}$
3	horizontal (+x) vertical (+y)	- yaw pitch	d_{yaw} d_{pitch}

Table 8.10: CosmoPlayer viewport control

Mouse Alone

The CosmoPlayer VRML viewport control paradigm (shown in Table 8.10) was slavishly copied. Modes 1, 2 and 3 refer to whether the left, centre or right mouse button is being held down respectively. The algorithm to implement this control paradigm is presented below;

STEP 1. Initialise cumulative transformation matrix C.

This represents the initial world $\rightarrow$ camera coordinate transform.

REPEAT

STEP 2. Read Mouse

Derive $x_{translate}$, $y_{translate}$, $z_{translate}$, d_{yaw} and d_{pitch} as CosmoPlayer

STEP 3. Create an incremental rotation matrix R based on Euler Yaw, Pitch and Roll angles.

$$R = R_x(d_{yaw}) \times R_y(d_{pitch}) \times R_z(d_{roll})$$

$R_i(A)$ stands for rotation round the i-axis by an angle of A.

STEP 4. Create an incremental translation matrix T

$$T = \begin{bmatrix} 1 & 0 & 0 & x_{translate} \\ 0 & 1 & 0 & y_{translate} \\ 0 & 0 & 1 & z_{translate} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

STEP 5. Update cumulative transformation matrix C

$$C' = T \times R \times C$$

STEP 6. Draw Scene

END REPEAT

Mouse + Headset Orientation Sensors

STEP 1. Initialise viewpoint position (v_x, v_y, v_z).

STEP 2. Initialise cumulative mouse rotation matrix C_R to the identity.

STEP 3. Initialise cumulative transformation matrix C.

This represents initial world $\rightarrow$ camera coordinate transform.

REPEAT

STEP 4. Read Mouse

STEP 5. Derive $x_{translate}$, $y_{translate}$, $z_{translate}$, d_{yaw} and d_{pitch} as CosmoPlayer

Note if mouse is not currently providing input, all 5 variables have value 0.

STEP 6. Form 4-vector d

$$d = C^{-1} \begin{bmatrix} x_{translate} \\ y_{translate} \\ z_{translate} \\ 0.0 \end{bmatrix}$$

The first three components of d give the incremental translation

$$\begin{bmatrix} v'_x \\ v'_y \\ v'_z \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} + \begin{bmatrix} d[1] \\ d[2] \\ d[3] \end{bmatrix}$$

STEP 7. Create absolute translation vector

$$T = \begin{bmatrix} 1 & 0 & 0 & v'_x \\ 0 & 1 & 0 & v'_y \\ 0 & 0 & 1 & v'_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

STEP 8. Create an incremental rotation matrix R based on Euler yaw, pitch and roll angles.

$$R = R_x(d_{yaw}) \times R_y(d_{pitch}) \times R_z(d_{roll})$$

STEP 9. Update cumulative mouse rotation vector C_R

$$C'_R = R \times C_R$$

STEP 10. Measure yaw, pitch and roll angles of headset orientation sensors.

STEP 11. Generate headset rotation matrix H

$$H = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times R_x(roll) \times R_y(pitch) \times R_z(-yaw) \times \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The pre- and post-multiplication is due to different Euler angles conventions in use.

STEP 12.

$$C' = C'_R \times H \times T$$

STEP 13. Draw Scene

END REPEAT

These algorithms were painfully debugged stage-by-stage and they were not stumbled upon by introspection. This is why they have been carefully documented here. The resulting effect is useful:

both mouse input and headset orientation can be used to navigate about virtual space intuitively. It was noticed that even CosmoPlayer gets the maths wrong just using a mouse for input, exhibiting spurious non-intuitive navigation at times similar to the un-debugged code. The experience of testing shows that while it is not noticeable if the maths is slightly wrong, the moment the maths becomes fully correct the navigation modality snaps into place as instantly being more intuitive. The reason the initial algorithm could not be changed into the second easily is because the mouse provides incremental relative navigation, whereas the headset provides absolute values. It was realised (painfully) that a total rewrite rather than an adaptation was required. It proved necessary to keep many more state variables in the second algorithm, to factor out the different rotations and translations originating from the different devices.

8.7.2 OpenGL and Projection Mathematics

The OpenGL projection matrix and the "pure" mathematical one are different in format, so they are fundamentally incompatible. The z value produced by the OpenGL projection matrix is specially scaled for use in depth buffering. The real world has no such requirement. Although incompatible, the OpenGL and real-world matrices can be chosen to give the same visual projection effect, which is all that is required. A working solution was reached by again going through a semantically intelligible OpenGL projection function, namely `glFrustrum`. The solution is documented below.

The mathematical projection model is:

$$P = \begin{bmatrix} \frac{f}{p_w} & 0 & u_c & 0 \\ 0 & \frac{f}{p_h} & v_c & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where f is the focal length; p_w is the pixel width; p_h is the pixel height; and (u_c, v_c) is the optical centre of the camera in pixel coordinates on the image plane. The OpenGL projection matrix **can** be arbitrary but a single function glFrustrum gives **the** interface for realistic 3D projections.

`glFrustrum(l,r,b,t,n,f)` produces a projection matrix:

$$P' = \begin{bmatrix} \frac{2n}{r-1} & 0 & \frac{r+1}{r-1} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

where l is the left edge offset on the projection plane; r is the right edge offset on the projection plane; b is the bottom edge offset on the projection plane; t is the top edge offset on the projection plane; n is the z-coordinate of the near clipping plan; and f is the z-coordinate of the far clipping plane. P and P' are different animals. The unintelligibility of P' implies that the only sensible approach is to produce a projection matrix using `glFrustrum` with arguments derived from P. Setting up the equations:

$$\frac{f}{p_w} = P(1,1) \quad \Rightarrow \quad p_w = \frac{f}{P(1,1)}$$
$$\frac{f}{p_h} = P(2,2) \quad \Rightarrow \quad p_h = \frac{f}{P(2,2)}$$

The near plane ($z = n$) of GL acts as the focal plane ($z = f$) of the real-world. However, only the ratio of f to pixel width is known. So we set f to something suitable as a near clipping plane (actually application dependent) and the size of the pixel on the image plane is then scaled accordingly. Similarly, we choose a suitable far clipping plane for our application. Code to convert from matrix P to an OpenGL projection is provided in Figure 8.8. There is an essential but inexplicable swap between `top` and `bottom` for the code to work; this has to be taken as another OpenGL funny. Note that the results which come out of VPS provide quite an asymmetric viewing frustum *i.e.* the optical centre is not in the centre of the camera's CCD. In fact, if one assumes that the optical centre is central then the resulting registration can be observed to be very bad.

Conclusion

A real-world projection (4 degrees of freedom) **cannot** be automatically translated into a OpenGL projection (at least 6 degrees of freedom) without some application knowledge. The problem is insoluble. However, practical solutions can be found as it is quite feasible to guess suitable values. The code presented in Figure 8.8 does appear to work. An OpenGL matrix (representing a genuine 3D pro-

```
double w, h, f, uc, vc, pw, ph;
double left, right, bottom, top, near, far;
double top2, bottom2;

get_window_width_and_height(&w,&h);
f  = 0.5;
uc = P(1,3);
vc = P(2,3);

pw       =     f /  P(1,1);
ph       =     f /  P(2,2);

// edges of projected image on image plane
left     =   - ( ( uc - 0.0 ) * pw ) ;
right    =     ( ( w  -  uc ) * pw ) ;
bottom   =   - ( ( vc - 0.0 ) * ph ) ;
top      =     ( ( h  -  vc ) * ph ) ;

// near and far clipping planes
near     =     f;
far      =     20.0;

// vertical swap is necessary - why ?
top2     = -bottom;
bottom2  = -top;

glFrustum(left,right,bottom2, top2 ,near ,far);
```

Figure 8.8: Deriving OpenGL projection from projection matrix P

jection) can, however, be translated into the real-world form *i.e.* the inverse mapping is well defined. A totally arbitrary OpenGL projection is probably capable of powerful and unexpected actions.

8.8 System Integration

Once the OpenGL browser had been developed and enhanced to provide stereoscopic output, the next step was to attach the following input peripherals:

1. VR-headset

2. video camera as position and orientation device (courtesy of VPS software)

Though the GPS position system was effectively integrated through the development of the above (because it can connect though the same positional device interface) it was **not** tested because the GPS only works outdoors whereas the remainder of the equipment is confined to indoors, through the necessity for a mains supply and the finite length of cabling. Clearly, to take this work further untethered operation is essential.

The integration of VPS and OpenGL both debugged the VPS (the orientation sub-matrix being the most notable) and tested out the software that converts VPS to OpenGL. The latter still has two fudge factors that cannot be explained, but before the debugging there were six such fudge factors distributed about the code. The mapping established between VPS and OpenGL is robust in operation. The software developed has a VPS-polling main loop that outputs to OpenGL. There is no interrupt servicing.

An important integration was to use the VPS input to drive the OpenGL browser which was previously driven by a mouse and/or headset orientation sensors. This has an X main loop to deal with interrupt servicing (necessary to accept mouse input) and optionally polls a relative orientation device (measurements are taken from the VR headset angular sensors). In theory it should be simpler to integrate an absolute positioning system rather than a relative one, as no merging of multiple device inputs is required. The symptom of the last integration bug was that one of the columns in the left eye image collapsed. This was so peculiar and incomprehensible that it was thought insoluble. However, this was eventually traced down to the fact that a GLX context seems to work only on a per-window basis (though its arguments suggest otherwise). The application uses multiple GLX windows, through the author's own implementation of the GLUT library. The latter had to be changed to support a context per window. Other final tune-ups include:

1. Mapping model coordinate system sensibly to VPS coordinate system

   This makes sure the physical floor corresponds to the virtual floor rather than the virtual wall *etc.*. This is on a per-model basis and involves a fixed transformation. For example, the mapping between the VPS coordinate system and the virtual VASE laboratory model coordinate system is already known because the VPS targets are at known positions within the physical VASE laboratory.

2. Matching computer and human eyes

   (a) Adjusting field of view (initially a bit wide angle) of OpenGL browser software to match that of VR headset.

   (b) Adjusting stereo disparity (initially that of hammer-head shark for clarity of effect) of OpenGL browser software to match that of a human.

scenario	positional input	orientation input
(1)	mouse	mouse
(2)	mouse	VR headset ($\pm$ mouse)
(3)	VPS	VR headset ($\pm$ mouse)
(4)	VPS	VPS ($\pm$ mouse)

Table 8.11: Supported combinations of input device

Both 1 and 2 were achieved by allowing the appropriate controlling parameters to be supplied on the command line and by comparing the real view in the VASE laboratory with the superimposed virtual view through the headset. After a few iterations it is possible to get the real and virtual views to coincide. Under command line control the OpenGL browser can accept various combinations of input device as shown in Table 8.11. A particular feature, is that two orientation devices can be used together. Though it is initially counterintuitive for two devices to provide the same information (like simultaneously trying to modify a shared variable in parallel programming) the set-up is useful. In scenarios (2), (3) and (4) the mouse can be used to change the relative alignment of the virtual world. This is useful to minimise rotating on one's chair, say, to gain a desired viewport in a VR application and useful to set-up alignment in an AR application. Scenario (4) is the indoor AR system that was successfully prototyped and demonstrated. Good registration of the real and virtual worlds was achieved. In fact, the quality of registration was better than the quality of the virtual model, as small discrepancies could clearly be observed between the relative sizes of doors and wall-spaces in the real world and the virtual model.

8.9 Future Visualisation

There is some support within OpenGL for some specific stereo hardware *e.g.* CrystalEyes stereo on the Silicon Graphic platform. However, there is no such support for the line-interleaving approach essential for the i-glasses. The proper and permanent solution is to put a device independent API in OpenGL that would support all stereo display types on as many types of hardware as possible. If time were available, this task would have been undertaken. The type of stereo that could be validly supported within OpenGL under the same API are.

1. Double scan-rate. The refresh frequency of the monitor is doubled (or increased as far as is possible). Within the area of a stereo window, the right eye and left eye images are shown on

alternative refreshes. A set of LCD shuttered glasses is worn which exposes alternate eyes to alternate frames. This provides a relatively inexpensive, but non-immersive stereo experience: the stereo scene is only present withing the confines of the edge of the monitor. There may be problems with brightness and visible flicker. Clearly there must be system code that responds to interrupts at the refresh rate to switch between the right and left eye images. The syncing for the glasses generally comes off the cable connecting the computer's video output to the monitor. LCD projection devices cannot currently refresh quickly enough for this effect to work in most such projective systems, which is unfortunate as the result would be considerably more immersive. The extra software required to support this under OpenGL is minimal and hardware specific.

2. Line interleaving. The video signal for the i-glasses requires the right and left eye images on alternate scanlines. The software development work described in this chapter, tells how this was achieved under the Linux operating system. The extra software required to support this technology under OpenGL is minimal and graphic-card specific if efficiency is required. The hardest part of the operation, for a software engineer who did not write the original code, is to find the few lines of code that require changing.

3. Anaglyphic. By combining the green component of the right eye's grayscale image and the red component of the left eye's grayscale image, and by wearing correspondingly tinted lenses over the two eyes, a stereo effect will be produced. This is called a pure anaglyph. No chromatic information is preserved. There are more advanced grayscale anaglyphs where only the disparity information is colour-coded to determine whether it is right-eye or left-eye. Without glasses this is reasonably viewable as a black and white photo; with glasses the brightness is better than with a pure anaglyph. Even more advanced are colour anaglyphs: not only is the disparity information colour-coded but some colour information is present in the image in an attempt to preserve some chromatic information in the original scene. Without glasses this is reasonably viewable as a colour photo; with glasses the brightness is better than with a pure anaglyph. This effect is achieved at the expense of some cross-talk or bleeding of the two images *i.e.* a faint shadow of the right image will be seen by the left eye and *vice versa*. There are a variety of schemes for colour anaglyphs of varying degrees of success. Opinion varies as to the value of the chromatic information, but the cyan-red colour variant is now the most widely

used anaglyph. The extra software required to support anaglyphs under OpenGL is small, but care needs to be taken regarding the performance of the colour processing, which will probably have to be done in software. This is an exceptionally good lowest common denominator stereo as practically all workstations are capable of displaying the 24-bit colour necessary to support it.

4. Lenticular display. By placing a vertically-oriented lenticular sheet over a TFT (Thin Film Transistor) display, and organising pixels in odd columns to be taken from the left-eye image and pixels in even columns to be taken from the right-eye image, then a stereo effect can be produced. Such a product has become available commercially for the first time from a firm called Dresden 3D [78]. The basic system is enhanced to track the positions of the pupils of the viewer's eyes, and to change the display accordingly. This overcomes the traditional disadvantage of stereoscopic systems of providing a single viewpoint, but naturally the tracking solution only works for a single viewer. The tracking is only appropriate for dynamically computable computer-generated images, not prerecorded live action sequences.

5. Autostereographic rendering. It is possible to provide an autostereographic (also know as "magic eye") rendering on an OpenGL scene. This is distinguishable from the previous approaches as the rendering technique is totally different: the depth buffer is involved rather than the right and left eye images. Some chromatic information may be retained in the scene [79]. Note that 3D graphics hardware is not capable of achieving the final autostereographic rendering of the scene, and this would have to be done in software.

6. Pulfrich effect. The Pulfrich effect is rather different to all the above and is included more from interest and completeness as supporting it needs no additional OpenGL software effort. By constant lateral movement of the viewpoint of a 2D projection of a 3D scene in one direction, and by wearing a darker lens over the appropriate eye a 3D illusion is created. The images may be sourced from specially choreographed live action filming (difficult) or produced by computer graphics (easy). As the trajectory of the viewpoint is greatly constrained, this technique is totally inappropriate for AR and only of limited use for VR. The dark lens inexplicably delays the brain's processing of that eye's image. The movement thus creates a virtual lateral displacement, which essentially provides a left and right eye image from a single source. Typically, the viewport will rotate about a scene constantly to keep the relative movement continual. The

Sequence	Model	Visible to Application Software	System Starting Point
1 (high level)	VRML	Perl + 'C'	FreeWRL
2	VRML	Java + 'C'	In-House Shared Virtual World
3	OpenGL	Tcl/Tk + 'C'	Togl
4	OpenGL	'C'	GLU
5	OpenGL	'C' + X	GLX
6 (low level)	OpenGL	'C' + X + hardware registers	Mesa H/W Drivers

Table 8.12: Successive implementations of AR system

strength of the effect is dependent on the speed of the movement and disappears totally when motion stops. In fact this is ideal for computer generated images as it is relatively trivial to ensure that this constant movement is present. Filming real-live scenes to take advantage of the Pulfrich effect is considerably more complicated. The Pulfrich effect has been little used for computer visualisation, and could probably be useful for certain applications, such as the visualisation of molecules where rotation is used to present structure already *i.e.* the kinetic depth effect could be turned into a true stereoscopic one.

A static photographic image equivalent of the software required called "The Stereoscope Applet" [80] was located on the Web. It has the flexibility to render stereo photographic pairs in almost any way imaginable, and its platform-independent utility is the best argument the author has witnessed for Java. The software works with JPS images (JPEG Stereo which is an ordinary JPEG file with the right image at the left and the left image at the right as if arranged for crossed-eyed viewing!).

8.10 Conclusion

Successful visualisation for AR was achieved *i.e.* the real and visualised virtual worlds were accurately aligned. There was considerable effort on the software front to ensure that the only limit on performance is the underlying 3D graphics hardware. It can be observed that despite all attempts to keep the software at as high a level as possible, it slipped to a lower level on no less than five separate occasions (see Table 8.12). This proved necessary for the following reasons:

1. to simplify code development
2. to obtain the necessary functionality

3. to circumvent bugs at a higher level
4. to obtain the necessary performance

Though at each stage much of the proceeding code was re-used, the narrative is an indictment of the current state of software engineering, and hard evidence against the practically of software re-use, which is presented at least in courses on software engineering as viable. Tanenbaum [81] wisely remarks: “If the hardware has an extremely efficient way of doing something, it should be exposed to programmer is a simple way and not buried inside some other abstraction”. His example is that a powerful RasterOp call should be visible to the programmer: the absence of RasterOp visibility is precisely the problem the author experienced. The difficulty also undoubtedly arose from the non-standard nature of the input and output devices used in AR, for which there is little prior art and for which existing software is not prepared. This suggests a virtualisation of many additional things as standard on computer systems that are not currently presented thus *e.g.* 3D displays and positional input devices. Support for positional and orientation devices is required, to which viewports and objects within a scene can transparently connect through well-defined software interfaces. It should be possible too for 2D devices to emulate such 3D devices, in standard and definable ways when genuine 3D hardware is not present.

Chapter 9

Conclusions

9.1 Challenges for AR

The principal observation from the experience gained in designing and building Augmented Reality (AR) systems is the immaturity of the discipline. This is discernible in several ways:

- lack of application software
- lack of hardware platforms for AR
- lack of peripherals suitable for AR
- lack of software structures to support AR applications

The current deadlock is that applications will not be produced until low cost and effective supporting technologies are in place. Yet, the drive to create the technology is absent due to the lack of these applications. Currently practical AR research spring-boards off computer games and VR technologies: the latter themselves are hardly mature. However, AR has unique additional demands that are not satisfied by current off-the-shelf components. The aim of this project was to break this deadlock and to move the domain forward by aiming at a particular application based on virtual archæological reconstruction *in situ*, with a particular emphasis on positioning technologies. The following sections examine each deficiency to see how it has been addressed.

9.1.1 Lack of Application Software

The archæological reconstruction application is an obvious demonstrator with the visual appeal and utility to promote AR. Software was developed to display stereoscopic representations of Roman buildings within a VR headset, the viewport being controlled by the location and orientation of the user's head. The basic core functionality of the application was achieved.

9.1.2 Lack of Hardware Platforms for AR

AR goes hand-in-hand with the physical freedom to move in one's environment, so there is a dependency on wearable computer technology. A wearable computer has been developed at Essex University to support the archæological and other mobile applications. It was not powerful enough to support the interactive 3D graphics required so all development and testing was done on desktop machines. The resulting lack of mobility limited the AR system to indoors (*i.e.* tethered) operation rather than outdoors as planned. A next generation of wearable has subsequently been developed at Essex University (Pentium-based instead of Cyrix 586-based) that has much improved 3D rendering capability.

The major technical problem in developing wearables is battery life. The more compute power required the faster the batteries become exhausted: power consumption goes as the square of processor speed. Commercial lightweight portable computers such as PDAs offer limited processing power. Newer process technologies; newer processor architectures; and newer power management strategies hold out the promise of sufficiently powerful wearable computers. Currently, the processing requirement of general AR outstrips the power of wearables. Research and prototype systems can to some extent be cut down to fit the available hardware. However, the underlying assumption behind the author's work is one of technology intercept. In an important sense it is immaterial how this mobile processing power is ultimately supplied *e.g.* the wearable could connect to powerful compute servers via high-bandwidth wireless communications.

9.1.3 Lack of Peripherals Suitable for AR

It is the view of the author that this is the most significant obstacle to AR. While VR headsets are necessary evils to present visual augmentations, measuring the position and orientation of the user's head to the appropriate accuracy is a problem. It is impossible to buy an off-the-shelf general purpose

device returning position and orientation, and so the majority of effort was dedicated towards obtaining this information. Software was written to locate the user's head using both vision processing and GPS technology. Software was also written to obtain the orientation of the user's head using both vision processing and (more trivially) the built-in orientation sensors in the VR headset. The positional accuracy obtained from the outdoors GPS system (1.7 m) was lower than the accuracy of the vision system developed (0.08 m). The angular accuracy of the vision system is around 1°. The quality of indoors registration obtained between the virtual and the real world was excellent and easily exceeded the quality of the virtual model itself. The caveat is that the latency of registration was high at around 1 second. No attempt was made to reduce this latency (despite the known techniques) as it was felt to be outside the remit of the project (see next section).

9.1.4 Lack of Software Structures to Support AR Applications

As AR is by definition interactive, it demands a real-time architecture. AR may have high input bandwidth in addition to the high output bandwidth inherited from VR. It is critical that the latency from input to output is minimal, and that the responsiveness of the system is at a maximum. A traditional operating system such as that used (Linux) is not suited to this architecture. There are nonetheless two claims in defence of the decision to use Linux:

1. Linux as development platform. Linux was only ever regarded as the development system; the target system should be real-time. Linux has significant advantages as a development platform due to the variety and quality of open-source software available.

2. Real-time versions of Linux are available. It is an open question whether the standard versions of Linux will become real-time. In the opinion of the author this is inevitable as today's advanced extensions become tomorrow's mundane incorporations. It may be that the transition from the development to target operating system happens passively through natural upgrades.

The acknowledgement of a potentially different target operating system is why the latency issues were not addressed. In an ideal operating environment, the latency should be no more than the cycle time *e.g.* the vision-based AR system (the most computationally intensive) works at 4 Hz on a 266 MHz processor, so the latency should be no more than 0.25 Hz. In fact, during the development period of the software, the latency of the video frame grabbing kernel drivers was known to have been considerably reduced, though the kernel of the development machine itself was not updated with these changes. The

software framework for an AR application is more than just having an underlying real-time operating system. Components within an AR application can clearly be identified: input devices (critically including positional input devices); esoteric display devices (including stereoscopic); virtual models (in various formats); and users (potentially distributed) over a network (wireless or otherwise). It should be possible to drop in any type of positional device; display device; or model format and indeed any number of users without changing the application software. This demands a flexible software architecture with well-defined interfaces for easy component integration.

The real-time software architecture necessary for an AR application finds its parallel in the real-time framework necessary to provide services on a wearable computer. Such an architecture for the latter has been fashioned at Essex, and is called "Sulawesi" [82] and this could be the supporting low-level layer for the AR framework. Suitable mechanisms for human computer interaction on a wearable are completely different from those on a desktop machine. Pre-emptive speech input and pro-active behaviours (say prompted by location) are desired capabilities of a wearable's system software; Sulawesi provides prototypes for both.

The difficulties of using a 3D positional input device to control existing 3D graphic visualisation packages to render in stereo is illustrated by the fact that the visualisation architecture had to be rewritten 6 times before a suitable platform was found! The only difference in the final platform (an OpenGL implementation) is that the bugs present were not catastrophic and could be coded around. Conventional workstation software is so geared into the notion of a keyboard and mouse that this is deeply ingrained even to the heart of applications that are fully 3D in scope. The abstraction of a 3D-service is obvious but unexploited and unimplemented. Flexible AR software should be able to connect multivarious 3D services to avatars and user viewports within a shared virtual/augmented world. Such a flexible approach gives the capability of merging multiple physical spaces (instead of the one of traditional AR) with one virtual space. To illustrate, it would be possible to place two football teams into two respective stadia, but using AR technology to make them appear to be playing (to both spectators and participants alike) in the one space. Such scenarios are increasingly defying categorisation. This would a be tribute to the flexibility of the enabling system architecture, rather than a condemnation of academic taxonomic schema. Is the game occurring in a single virtual space, in two physical spaces, or in two augmented spaces?

Note that it would be possible for every individual concerned to have a different view of the game: perhaps the avatars representing the opponents could be different for every player. This application

could demand anything up to full-motion capture for each player. This goes beyond the simple head location and orientation capture of the system actually developed: a viewport is all that is required to display an inanimate object like a building. However, this more complex application is described to indicate the capabilities that an AR framework should support now for future compatibility. The crime of ignoring future expandability and generality is as great as hardwiring mouse-specific code within the depths of a 3D application!

The pros and cons of using different software approaches are described in Chapter 8 from the genuine experience of trying all these out. In the last resort, the visualisation software was essentially written totally from the ground up using only the most basic underlying libraries. Considerable time would have been saved if the author had written the system from scratch initially. This is a condemnation not just of the slightly specialised software that may not be adequately tried and tested, but of software as a whole. While a "convenience" library may be useful for doing simple tasks with limited functionality, the moment something a little more involved is required the layer below has to be entered, so the layer above becomes a millstone. This is a result of inadequate abstract design in interfaces. As long as such fundamental errors exist even in simple interfaces, it is difficult to imagine that a suitable software framework for AR could be available off-the-shelf. However, the aim of a software AR framework is a laudable one. Provided such a framework is in place and practical AR input devices are available, then the real work of writing AR applications can get underway. However, the demands that AR applications will make of an underlying framework are not yet well defined, and it is through application-directed research (such as has been carried out) that the requirements are becoming clearer. The basic functionality that is currently missing, is to attach 3D input devices flexibly to viewports and avatars in virtual worlds.

9.2 Technologies for AR

The author's belief is that the current most effective approach for determining position and orientation for outdoors AR applications such as Gosbecks is the use of multiple position determining technologies in combination, in particular the use of GPS, INS and vision processing. Vision processing provides the automatic initial registration. Indoors, the combination of INS and vision processing with an increased number of targets is appropriate. The findings from the principal component technologies investigated are discussed in the following sections.

9.2.1 GPS

At the start of this project GPS technology held out the greatest promise for outdoors AR applications. This is still true. In particular the switching off of the timing degradations on the non-military signals from GPS satellites on 1st May 2000, enables unprecedented positional accuracy ($\pm$ 4 m) anywhere on the earth's surface using a single handheld receiver. However, the positional accuracy required for AR is greater, and this necessitates the use of more than one receiver in the technique of differential GPS. Unfortunately, differential GPS systems are expensive. There are two approaches:

1. buy a proprietary multi-receiver system (expensive)
2. buy a single receiver (cheap) but subscribe to a service which radio broadcasts a live differential correction signal (expensive)

Two alternative models for differential GPS to support AR were developed and then implemented:

1. buy two receivers (cheap) but write one's own code (free but expensive in time) to process the raw data coming from these units.
2. write software (free) to broadcast a differential correction signal over the Internet starting with the raw data from a receiver (cheap).

Our home-coded differential GPS system achieved an accuracy of 1.8 m. This is still not good enough for AR. An analysis demonstrated the algorithm does the best it can with the quality of data at its disposal, so obtaining better accuracy demands better hardware or more advanced processing. A special metric was developed to show this. It determines the quality of a continuously varying quantity like a satellite range measurement, based on the average value of n when the n^{th} order difference breaks down in not being larger in magnitude than the $(n+1)^{th}$ order difference. This proved much more useful than trying to determine GPS quality from the nature of a final position fix.

The differential GPS hardware was delayed by two years. When the GPS hardware finally arrived, it was found to be broken (this was established beyond reasonable doubt over a period of 3 months) and user support for the manufacturers were contacted. No response or help was ever obtained. It was finally discovered through a GPS contact in Chile (though a GPS contact in Holland), that the version of the firmware supplied with the GPS equipment was faulty.

At this stage the secret proprietary protocols (holding data needed by differential calculations) of the ultra-cheap handheld G12 GPS receiver from Garmin had just been cracked by the Internet

community, so writing up the thesis (which should have started at this stage) was postponed for three months while the differential GPS system was ported, debugged, commissioned, evaluated and studied on top of consumer-grade hardware. Time constraints did not permit moving to a more advanced processing technique. Furthermore, the write-up was abandoned for a strict fortnight to implement the LAMBDA method of differential GPS processing. The LAMBDA code produces position fixes of centimetric accuracy using pre-existing data files, but time constraints prevented tests using data from the G12 units as the code first needs to be made robust to cycle slips. The LAMBDA method is a time-integrated one, so there is concern over its applicability to a dynamic AR scenario.

The recommended approach for AR is to have two (or multiple) GPS receiver systems at the site of the application: all factors are thus under local control instead of depending on the vagaries of a service provider. Wireless communications can be used to make the base station's data available to the user's wearable computer. In a location such as Gosbecks with clear views of the sky, differential GPS can work at a suitably high accuracy for AR. However, is this possible with low end receivers? More work is required to establish this. The capabilities of high end GPS (*i.e.* dual wavelength units with accurate and stable phase measurements) will ultimately move to single chips solutions. There is a requirement for public domain reference codes to give high end capability to all when such ASICs become available at low cost. There is also huge potential to use the Internet as the infrastructure to communicate differential GPS data, and to provide unprecedented accuracies through mass integration of GPS measurements.

9.2.2 Vision Processing

Fiducials were initially rejected as unsuitable for outdoors AR. However, it was eventually realised that a few targets at strategic points in any AR environment would usefully automatically calibrate the system. Eventually, as the impossibility of taking our AR hardware outdoors emerged and working GPS hardware did not arrive until the end of the project, vision processing became the primary technology for registering real and virtual worlds. The visual positioning system developed (or VPS) was the unexpected success of the project, because it resulted in a self-contained product of suitable accuracy for AR that has been released on the Internet *i.e.* there are no loose ends. Of course, many aspects of the VPS could be profitably enhanced, but it is good enough and its utter simplicity (this is certainly true of its final instantiation) gives it considerable robustness.

The VPS was never intended as a "research" system, but merely as a means to an end of produc-

ing AR research prototypes. However, despite not pushing the limits of vision processing research, the VPS does employ a number of novel techniques *e.g.* the novel target design makes locating and identifying targets a trivially simple problem. One early novel target design uses a shaded logarithmic spiral pattern to yield, under edge processing, a clear single spiral edge, which is identifiable by its high "winding number" (a curve integral). The design was suggested by the variable camera to target distance, because the only scale invariant shapes are logarithmic spirals and straight lines. A novel aspect of the final image processing is that the image is turned into a Region Adjacency Graph (RAG) almost straight away. All subsequent processing is graph-based and highly tractable: target detection, target identification and even noise removal. In fact the graph processing makes the choice of filter size which is often critical in image processing largely immaterial. The VPS can recognise targets that are both very large in the scene and targets that are very small.

The application of the Faugeras method to establish camera position and orientation from detected target positions within a 2D image and knowledge of those target positions within a 3D world is fairly standard textbook stuff. However, it is worth remarking that, on a frame by frame basis, the method does not produce particularly wonderful results. This is only visible if one is compositing off the optical axis of the camera. On the optical axis, the two matrices that result from the Faugeras factorisation, are recombined so the inexact nature of the factorisation does not become apparent. The average calibration matrix obtained over several frames is more reliable, and it is always recommended to determine a good quality calibration matrix in advance, prior to an interactive AR run. Of course, this is only possible with a fixed focal length camera, or with an automatically controlled zoom camera.

The design of the targets provides four "tie points" between the 2D image and the 3D world whereas traditional targets only supply one such point. To perform a Faugeras calibration and a position fix, 6 non-coplanar tie points are required. To perform a position fix with a known calibration matrix, 4 tie points are required. Consequently a position fix can be obtained with only a single target in view which is a considerable advance on other similar systems. Now if this target is distant, then the position fix may not be that reliable. However, by the time two targets come into view, the position fix will be rock solid. The VPS happily deals with coplanar targets: the special mathematical treatment of this case was worked out and the resulting positional accuracy, contrary to conventional wisdom, was essentially unimpaired. To illustrate the flexibility of the VPS, a 3D video joystick was developed. By manipulating a small cardboard cube, the user can intuitively supply position and

orientation information to an application.

An important aspect of the VPS, unlike similar systems, is that it relies on commodity- cheap processing platforms such as PCs, commodity-cheap frame grabbing boards and inexpensive A4 targets that can be printed out on a laser printer. In short, it is available to practically everyone. The combination of increasingly powerful PC processors and optimised image processing code, means that the VPS can do image processing and position determination at interactive rates. For example, the VPS runs at 4Hz on a 266 MHz PC and real-time operation on a commodity platform lies in the not too distant future.

9.3 Future Work

Targetless Visual Positioning System

Video mosaicing techniques can produce a seamless panorama from a sequence of images, or a high-resolution "plan" image of a planar surface. The established frame-to-frame geometric transformation, could be alternatively used to provide camera position and orientation for AR. The camera could point at the ceiling, say, and provided each locale is suitably unambiguous, no targets would be required. Otherwise, any distinguishable targets could be used without first surveying their positions. Gross alignment and orientation is establishable from a wide angle image; fine positioning is achievable from a lens of longer focal length. A special lens design may even be appropriate: there is no requirement for a conventional image.

Extend LAMBDA Code

The LAMBDA method software should be extended to deal with cycle slips and satellite outages. It will then be possible to establish whether the inexpensive G12 receiver from Garmin is capable of supporting this high accuracy GPS positioning technique. In particular, it should be tested at Gosbecks where there is full sky visibility, and where 12 satellites (rather than the 6 at Essex University) are typically visible.

A Full 6-Axis Accelerometer should be Obtained/Developed

This is essential to investigate GPS and INS (INertial System) integration. For AR the equipment should be as small and lightweight as possible. Software, if none can be found in the public domain,

should be written to integrate the INS measurements obtained with those of GPS.

Pass VRML models to AR Framework

It should be possible to use Virtual Reality Modelling Language (VRML) to supply models directly for AR. There are two approaches:

1. integrate VRML parser with existing visualisation architecture

   Such a parser is now available from the University of Washington.

2. hack open source VRML browser

GPS Operation with Fewer than 4 Satellites

Current GPS receivers give up when less than 4 satellites are visible. A number of possible techniques, including the use of hybrid technology, are suggested in Chapter 7 to circumvent this serious shortcoming which disqualifies the use of GPS in urban environments where typically only 2 satellites may be in view. It would be extremely valuable to thus extend the range of GPS into urban areas.

9.4 Concluding Remarks

Taking an overview of the project, a clear pattern emerges of moving existing technologies from other domains into the field of AR. To construct a system for personal AR involves tailoring (ultimately in terms of cost and weight) and recombining these technologies to obtain the appropriate levels of performance. The work for this project has started to bring GPS and vision processing into the AR fold. Only once components have been built based upon these individual or merged disciplines, and are presented in an easy to use form, is there the freedom to produce true AR applications. Once AR applications exist that people will want to use, then the real shake-out between technology and utility will occur.

The display technology for AR is seriously problematic as it is generally an encumbrance, and this work has not addressed the issue. The use of a VR headset for the duration of a tour of an archæological site is not considered excessive for what is an occasional experience. It may be that an in-car augmentation is the first mainstream take-up of AR technology, much in the same way that

in-car navigation systems are introducing GPS to the general public. Hitherto, users of GPS have generally been trekkers or marine-based with some existing concept of navigation.

Much of the requisite software is commercially sensitive, yet all the project work has been done with public domain codes and the outputs themselves had been made public domain. Given the technical complexities of such code, it is likely that the availability of such software though non-commercial agencies will occur only very slowly. The progress of GPS code onto the Web, has been observed to be slow. The current state of AR is too multi-disciplinary. This is not to say that multi-disciplinary skills are a bad thing, but to insist that all AR developers have these skills is counter productive. The mainstream trend is the movement of high-end aircraft navigation technologies, into vehicle tracking and this will ultimately move to personal tracking technologies. This trend has only been observable at the end of the project rather than at the beginning. As personal AR demands orientation information this is why it is a direct descendent of avionic technology, where orientation information is also vital. In the author's view there are three specific technological challenges to be met to open up the AR domain for all:

1. lightweight position and orientation sensors
2. public domain software to achieve high-end differential GPS performance
3. public domain software to hybridise GPS and INS

Successful AR is making the correct technological choices, and using the appropriate algorithms and software to integrate these technologies. The initial choices of technology for this project (GPS + INS) are still believed to be correct and this is clearer now than at the beginning. The additional requirement for robust computer vision software to achieve initial registration was identified during the project. No such public domain software existed at the time concerned, and the outcome of the effort expended here is generally useful and can support an indoors AR set-up without any other technologies being involved.

Augmented Reality is inevitably going to be the principal way in which humans interact with their computers whilst on the move: extra information will augment the user's view of the real world to assist them in their tasks. What physical form will this augmentation take, and what will AR applications do? These questions are unanswerable, because research cannot be prescriptive about the uptake of technology. AR is at the stage of struggling to providing the basic mechanisms that will be the building blocks of tomorrow's systems. Visual augmentation is particularly challenging, and

this work has produced a number of enabling tools which have resulted in a prototype AR system. It is noteworthy that all the technology used is inexpensive and “off-the-shelf”, and by placing the software in the public domain, these tools will be made available to the wider community to enable further work. One only has to look at the advances in output processing (in particular 3D graphics) even since Virtual Reality was originally hyped in the media to appreciate the similar leaps that are technologically inevitable for the input processing which characterises and distinguishes AR. Position determination is only a beginning, yet the leverage even here is considerable. Prediction is decidedly unwise, but permit the author a certain excitement and impatience.

Appendix A

Notes on Architectural Models

Some background information on the virtual models that were employed is presented here:

Gosbecks Temple Complex This consists of a glorious monumental square portico (or covered corridor) with sides around 100 m in length (Figure A.1). There are three sets of concentric foundations: the inner and middle foundations supported continuous rows of columns and the outer foundation supported the external wall. Within the portico was an area of special religious significance called the sanctuary. Access to the sanctuary was permitted only to priests and other religious functionaries, a policy enforced by a massive square ditch over 3.3 m in depth delineating the area. The sanctuary contained a Romano-Celtic temple, and probably alters and statuary. The capacity of the portico is estimated (at a squeeze) to be around 5,000. This is considered to be no accident given that the capacity of the adjacent theatre is also around 5,000. A virtual model of the Roman theatre has not been pro-

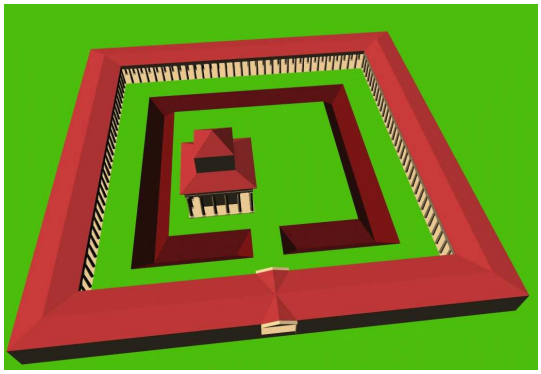

Figure A.1: Temple complex: aerial view

Figure A.2: Temple complex: ground-level view

Figure A.3: Temple of Claudius

Figure A.4: Colchester Castle

duced. Clearly, doing so, as well as modelling the gardens believed to have been at Gosbecks, would enrich the application. The temple complex model is one of the ones for the target augmented reality application at the Gosbecks site. The model itself is extremely large, consisting of a large number of columns each of which has a complex geometry (Figure A.2).

Temple of Claudius This Roman temple (Figure A.3), reckoned to be the first stone building constructed in Britain, used to exist in the centre of Colchester, on the site of the Norman castle (Figure A.4) that now serves as a museum. In fact, the model is of the second and grander temple dedicated to Claudius that was built on the same location. The first temple was destroyed by Boudica, the British Queen of the Icenii tribe, as she razed the town, in retribution for Roman atrocities against her family. The temple of Claudius must have be a spectacular sight, as it was entirely clad in marble and was even larger than the castle is now, which is impressive enough. The virtual model is considerably simpler to render than the one of Gosbecks, and it was used as the “test model” where high-power graphics hardware was not available.

VASE Laboratory This model was constructed of the Visual Applications and System Environment Laboratory at the University of Essex, where the development work was undertaken. As the model is not complex, it was used to investigate interactive methods of virtual navigation. It was also used to test the final AR system. Given that a real VASE laboratory exists, correspondence of real and virtual worlds could be directly verified using an indoor testbed.

Figure A.5: Physical VASE laboratory

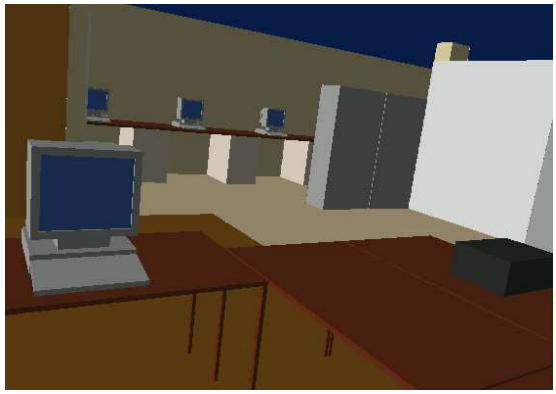

Figure A.6: Virtual VASE laboratory

Appendix B

Analysis of Low Level AllStar Data

B.1 Introduction

Whatever the complex procedure may be to convert the integrated carrier phase measurement of the AllStar GPS Unit (which is known to be different from the true integrated carrier phase) to a true measure of the carrier phase range, multiplying by the expected constant of proportionality ($\frac{\lambda}{1024}$) should give an indication of what is going on. The results were produced at both 10Hz and 1Hz were analysed.

The software to process the printout from the differential GPS system was written in *awk*. The software to fit straight lines to graphs was written in *octave*, which supports a MATLAB-like scripting language and produces output via gnuplot.

For the two AllStar receivers with a zero-baseline antennae set-up:-

$$\begin{aligned} P_{r_2} - P_{r_1} &\approx 0 \\ C_{r_2} - C_{r_1} &\approx 0 \end{aligned}$$

Consider a single receiver at two successive time intervals. The change in satellite range (as it flies overhead) should be the same.

$$\begin{aligned} P_{t_2} - P_{t_1} &\approx C_{t_2} - C_{t_1} \\ \textit{i.e.} \quad (P_{t_2} - P_{t_1}) - (C_{t_2} - C_{t_1}) &\approx 0 \end{aligned}$$

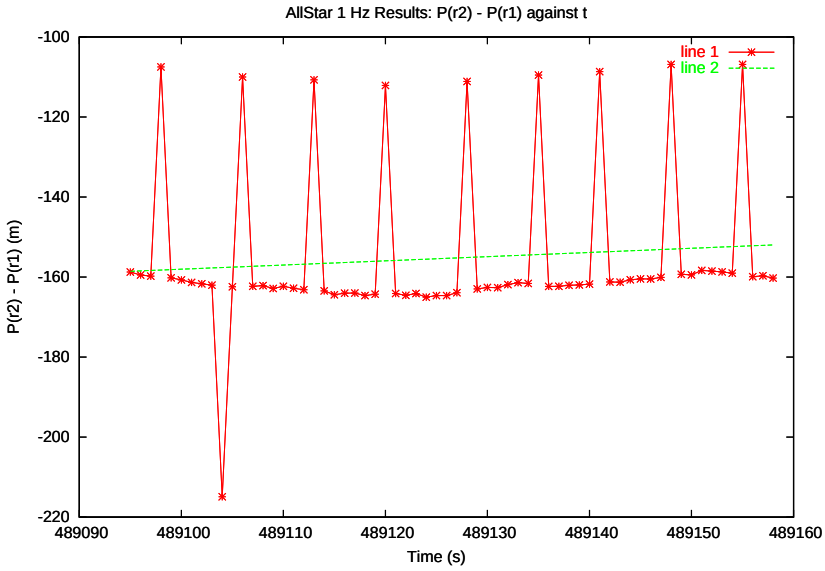

Figure B.1: 1 Hz AllStar raw data: $P_{r_2} - P_{r_1}$ against t — flat with blips

B.2 1 Hz Analysis

B.2.1 Results

1. $\boldsymbol{P_{r_2} - P_{r_1}}$ against $\boldsymbol{t}$ at 1 Hz

   Graph B.1 shows $P_{r_2} - P_{r_1}$ is essentially a constant (*i.e.* -160 metres). Quantisation effects/errors of some sort are visible as "blips" in the experimentally obtained data. This definition of "blip" will be used subsequently. One might expect (in an ideal world) the difference to be zero, but the figure presumably gives an indication of the accuracy to which the hardware is measuring the pseudorange. A linear fit to the data gives:

   $$P_{r_2} - P_{r_1} = 0.104664 \times t - 51349.098173$$

2. $\boldsymbol{C_{r_2} - C_{r_1}}$ against $\boldsymbol{t}$ at 1 Hz

   Graph B.2 is a straight line graph, with $C_{r_2} - C_{r_1}$ increasing linearly with time. $C_{r_2} - C_{r_1}$ should be constant! A linear fit to the data gives:

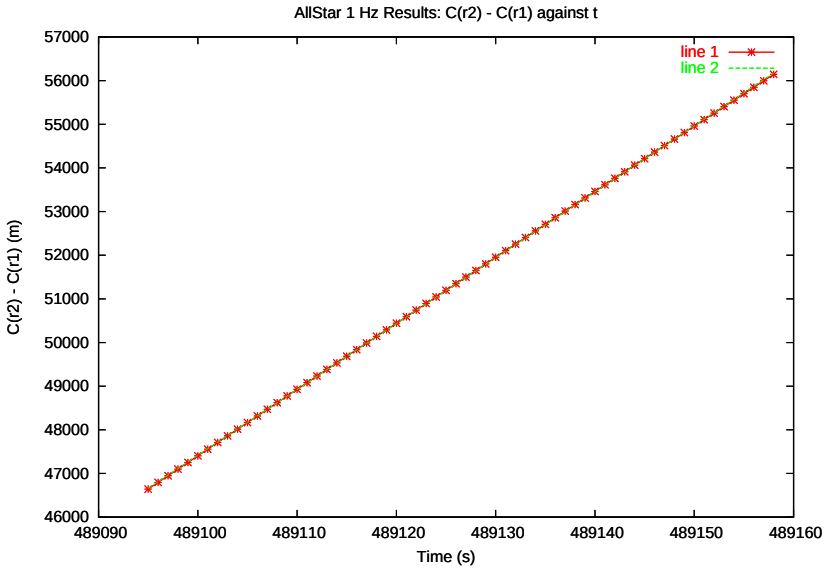

Figure B.2: 1 Hz AllStar raw data: $C_{r_2} - C_{r_1}$ against t — not flat as expected

$$C_{r_2} - C_{r_1} = 150.960714 \times t - 73787470.072298$$

3. $\boldsymbol{C_{r_1}}$ against $\boldsymbol{C_{r_2}}$ at 1 Hz

   Graph B.3 is a straight line. A linear fit to the data gives:

$$C_{r_1} = 1.504347 \times C_{r_2} - 95243.151595$$

   Surely the phase information should be in lock step?

4. $\boldsymbol{P_{r_1}}$ against $\boldsymbol{P_{r_2}}$ at 1 Hz

   Graph B.4 is a straight line of gradient 1. A linear fit to the data gives:

$$P_{r_1} = 1.000221 \times P_{r_2} - 4581.717999$$

5. $\boldsymbol{C_t}$ against $\boldsymbol{P_t}$ for $\boldsymbol{r_1}$ at 1 Hz

   Graph B.5 is a straight line of gradient 1. A linear fit to the data gives:

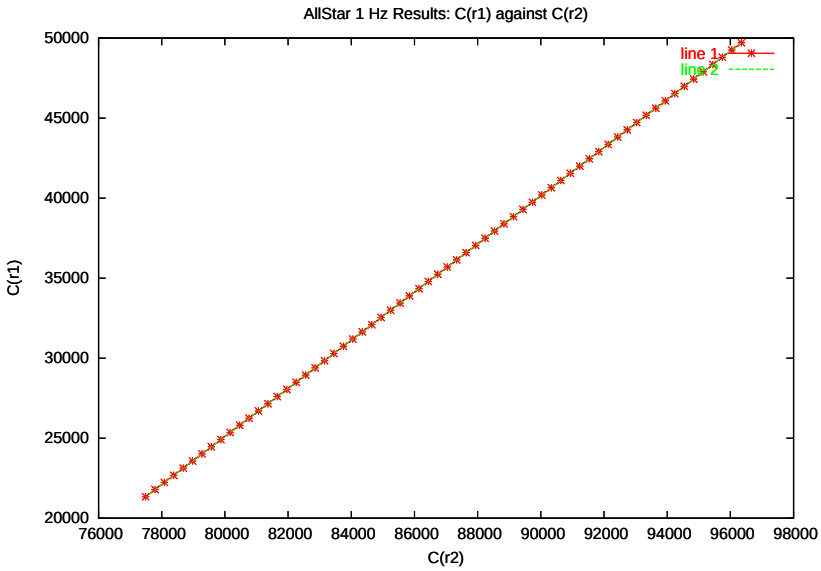

Figure B.3: 1 Hz AllStar raw data: C_{r_1} against C_{r_2} — gradient not 1 as expected

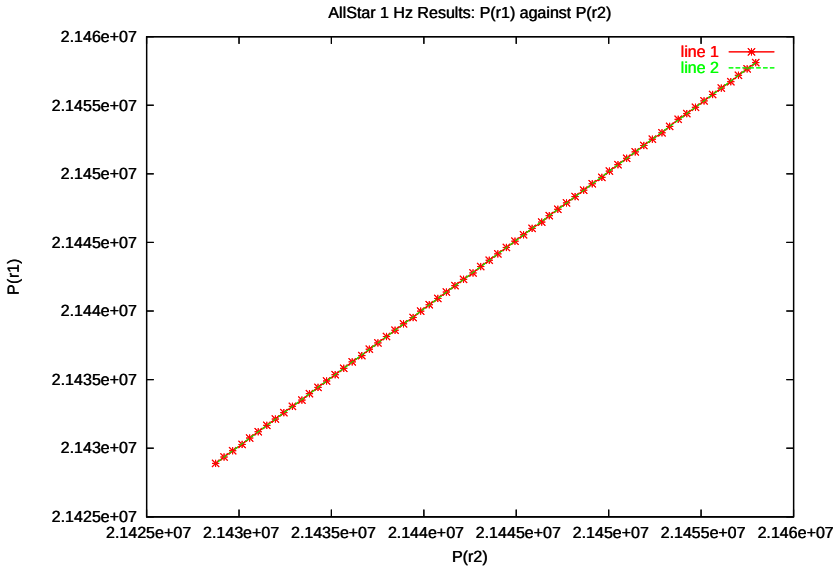

Figure B.4: 1 Hz AllStar raw data: P_{r_1} against P_{r_2} — gradient is 1

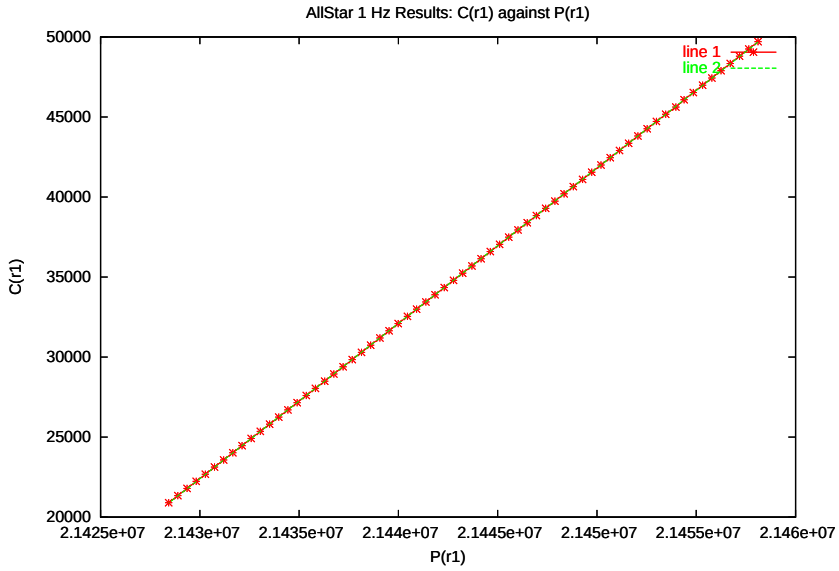

Figure B.5: 1 Hz AllStar raw data: C_t against P_t for r_1 — gradient is 1

$$C_t = 0.970672 \times P_t - 20779098.407675$$

The implication is that unit r_1 is behaving!

6. $\boldsymbol{C_t}$ against $\boldsymbol{P_t}$ for $\boldsymbol{r_2}$ at 1 Hz

   Graph B.6 is a straight line. A linear fit to the data gives:

$$C_t = 0.645388 \times P_t - 13752357.603493$$

   Why is the gradient not 1? This implies unit r_2 is misbehaving!

7. $\boldsymbol{P_{t_2} - P_{t_1}}$ against $\boldsymbol{t}$ for $\boldsymbol{r_1}$ at 1 Hz

   Graph B.7 is a straight line showing roughly small linear increase in first order difference of P. There is a single blip. A linear fit to the data gives:

$$P_{t_2} - P_{t_1} = 0.064828 \times t - 32172.888169$$

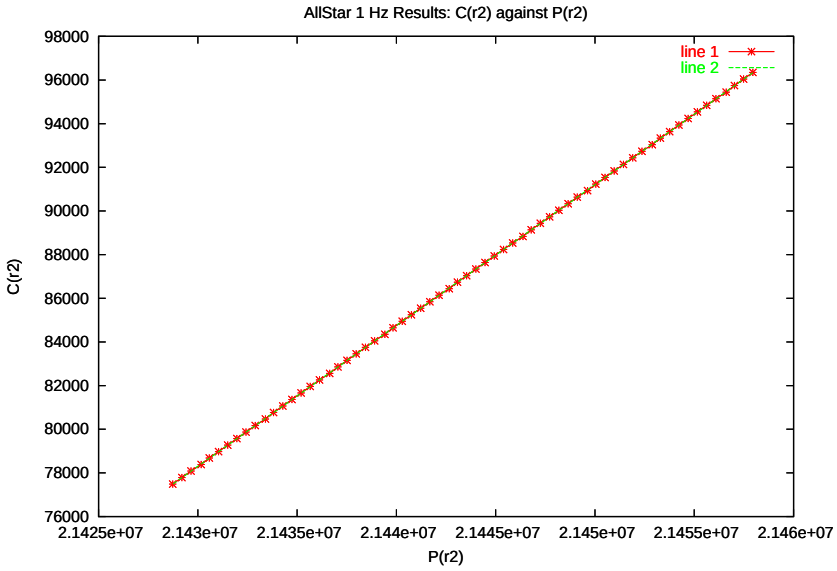

Figure B.6: 1 Hz AllStar raw data: C_t against P_t for r_2 — gradient not 1 as expected

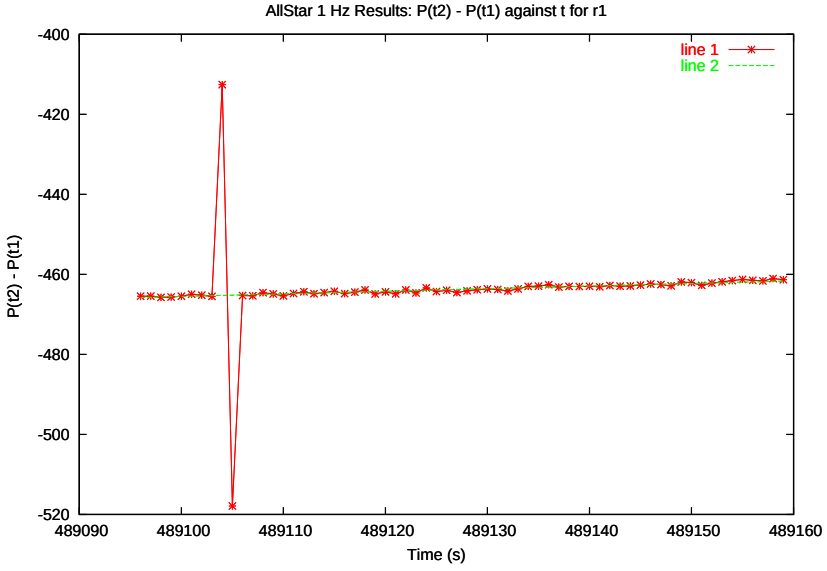

Figure B.7: 1 Hz AllStar raw data: $P_{t_2} - P_{t_1}$ against t for r_1 — flat with blip

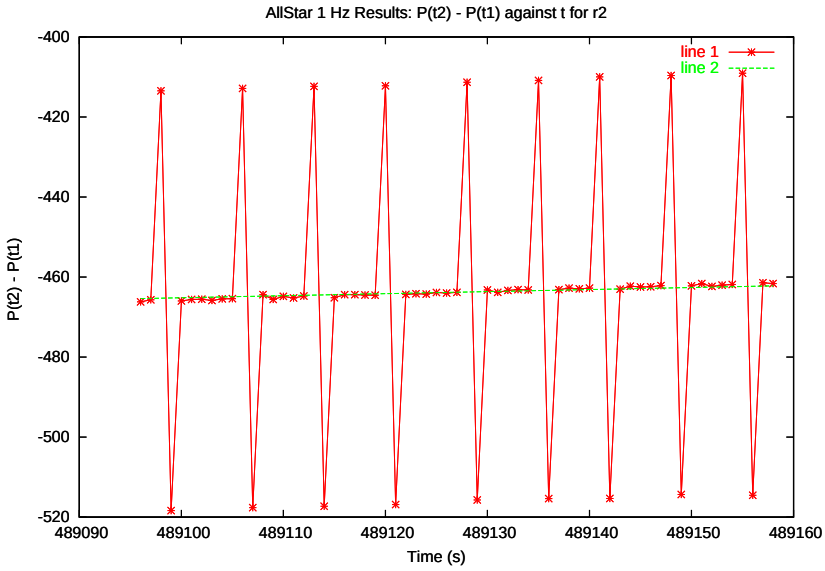

Figure B.8: 1 Hz AllStar raw data: $P_{t_2} - P_{t_1}$ against t for r_2 — flat with blips

8. $\boldsymbol{P_{t_2} - P_{t_1}}$ against $\boldsymbol{t}$ for $\boldsymbol{r_2}$ at 1 Hz

   Graph B.8 is a straight line showing roughly small linear increase in first order difference of P. There are more blips than for the other receiver. A linear fit to the data gives:

$$P_{t_2} - P_{t_1} = 0.051243 \times t - 25527.960086$$

9. $\boldsymbol{C_{t_2} - C_{t_1}}$ against $\boldsymbol{t}$ for $\boldsymbol{r_1}$ at 1 Hz

   Graph B.9 serpentines around a straight line of gradient 0.131995. Shows roughly linear increase in first order difference of C. A linear fit to the data gives:

$$C_{t_2} - C_{t_1} = 0.131995 \times t - 65012.255153$$

10. $\boldsymbol{C_{t_2} - C_{t_1}}$ against $\boldsymbol{t}$ for $\boldsymbol{r2}$ at 1 Hz

    Graph B.10 serpentines around a straight line of gradient 0.060515. Shows roughly linear increase in first order difference of C. A linear fit to the data gives:

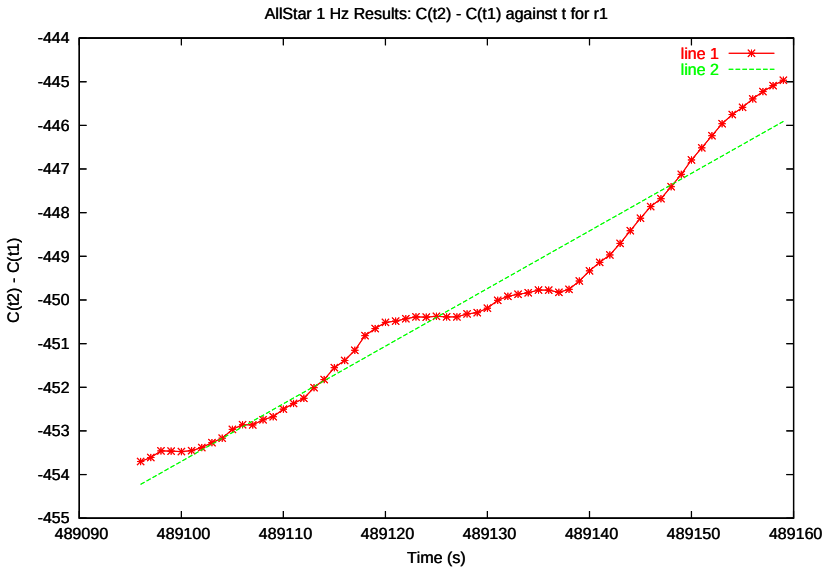

Figure B.9: 1 Hz AllStar raw data: $C_{t_2} - C_{t_1}$ against t for r_1 — straight line

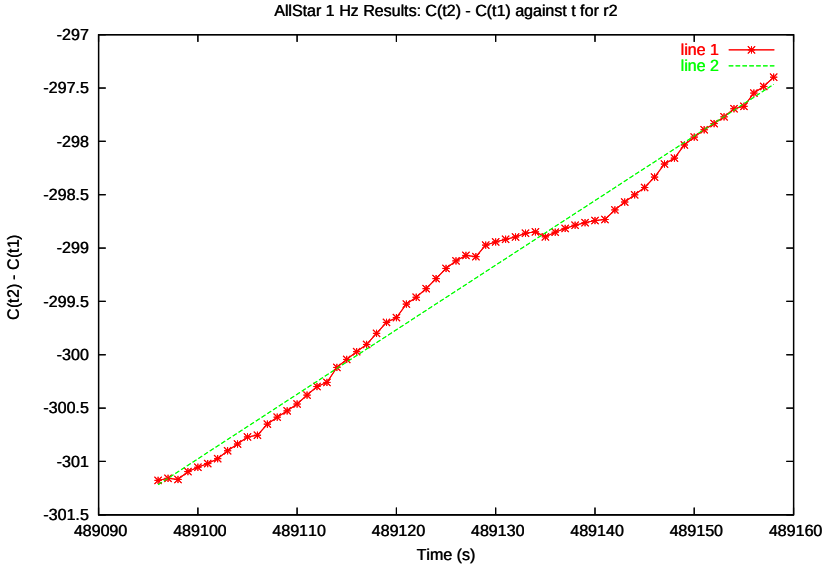

Figure B.10: 1 Hz AllStar raw data: $C_{t_2} - C_{t_1}$ against t for r_2 — straight line

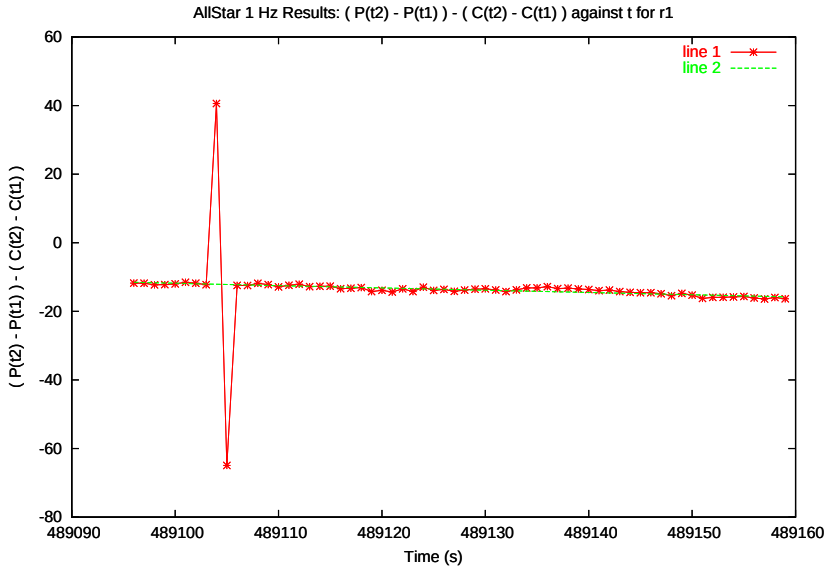

Figure B.11: 1 Hz AllStar raw data: $(P_{t_2} - P_{t_1}) - (C_{t_2} - C_{t_1})$ against t for r_1 — flat roughly zero

$$C_{t_2} - C_{t_1} = 0.060515 \times t - 29899.038781$$

11. $\boldsymbol{(P_{t_2} - P_{t_1}) - (C_{t_2} - C_{t_1})}$ against $\boldsymbol{t}$ for $\boldsymbol{r_1}$ at 1 Hz

    Graph B.11 is approximately a flat straight line (roughly constant value of zero) as expected. Decreasingly slightly with time, single blip. A linear fit to the data gives:

$$(P_{t_2} - Pt_1) - (C_{t_2} - C_{t_1}) = -0.067170 \times t + 32841.254925$$

12. $\boldsymbol{(P_{t_2} - Pt_1) - (C_{t_2} - C_{t_1})}$ against $\boldsymbol{t}$ for $\boldsymbol{r_2}$ at 1 Hz

    Graph B.12 (overlooking multiple blips) shows value is roughly constant. However, this value should be 0 not -160 metres. The indication is that the values of C are romping past those of P with constant acceleration! Again why more blips than the above. Is r_2 misbehaving? A linear fit to the data gives:

$$(P_{t_2} - Pt_1) - (C_{t_2} - C_{t_1}) = -0.009276 \times t + 4372.604758$$

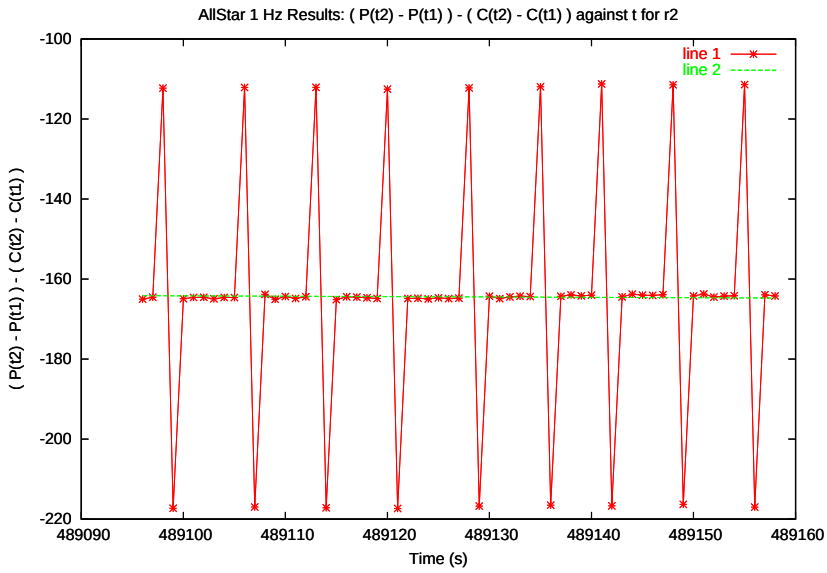

Figure B.12: 1 Hz AllStar raw data: $(P_{t_2} - P_{t_1}) - (C_{t_2} - C_{t_1})$ against t for r_2 — not 0 as expected

B.2.2 Conclusion

P from r_2 (many blips) is much less behaved than P from r_1 (a single blip). C from r_2 is changing at a rate $\frac{2}{3}$ that of C from r_1, whereas they should be be changing at the same rate. This implies the phase measurements from r_2 are completely wrong - certainly enough to throw out the differential GPS calculations. In short the problematic P from r_2 and the wrong C from r_2 may be symptoms of a problem with the unit. Certainly, the problematic P from r_2 would be enough to through off differential GPS calculations using P alone. Indeed, differential GPS using P alone is slightly worse than single receiver GPS. Differential GPS using P and C (which should be the most accurate set-up) is much worse than single receiver GPS. In short, something seems to be wrong with r_2.

B.3 10 Hz Analysis

B.3.1 Results

1. $\boldsymbol{P_{r_2} - P_{r_1}}$ against $\boldsymbol{t}$ at 10 Hz

   Graph B.13 shows $P_{r_2} - P_{r_1}$ is essentially a constant (*i.e.* -60 metres) with time though quantisation effects/errors of some sort are visible as "blips" in the experimentally obtained data.

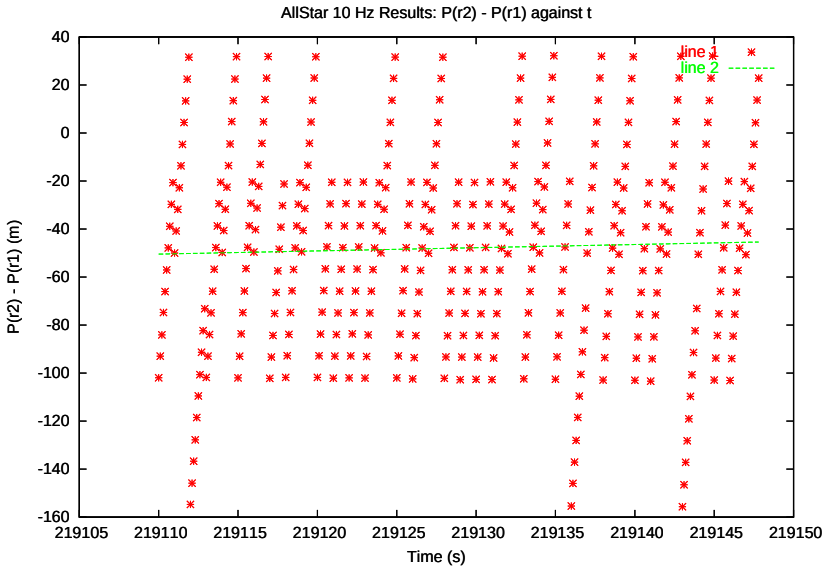

Figure B.13: 10 Hz AllStar raw data: $P_{r_2} - P_{r_1}$ against t — flat with blips

Suspiciously the sawtooth edges each consist of 10 points (which is the frequency) *i.e.* there are clear artifacts.

A linear fit to the data gives:

$$P_{r_2} - P_{r_1} = 0.133275 \times t - 29252.323041$$

2. $\boldsymbol{C_{r_2} - C_{r_1}}$ against $\boldsymbol{t}$ at 10 Hz

   Graph B.14 is a straight line graph, with $C_{r_2} - C_{r_1}$ increasing linearly with time. $C_{r_2} - C_{r_1}$ should be constant! Surely the phase information should be in lock step?

   A linear fit to the data gives:

$$C_{r_2} - C_{r_1} = 91.163203 \times t - 20089716.992975$$

3. $\boldsymbol{C_{r_2} - C_{r_1}}$ against $\boldsymbol{t}$ at 10 Hz

   Graph B.15 is a straight line. A linear fit to the data gives:

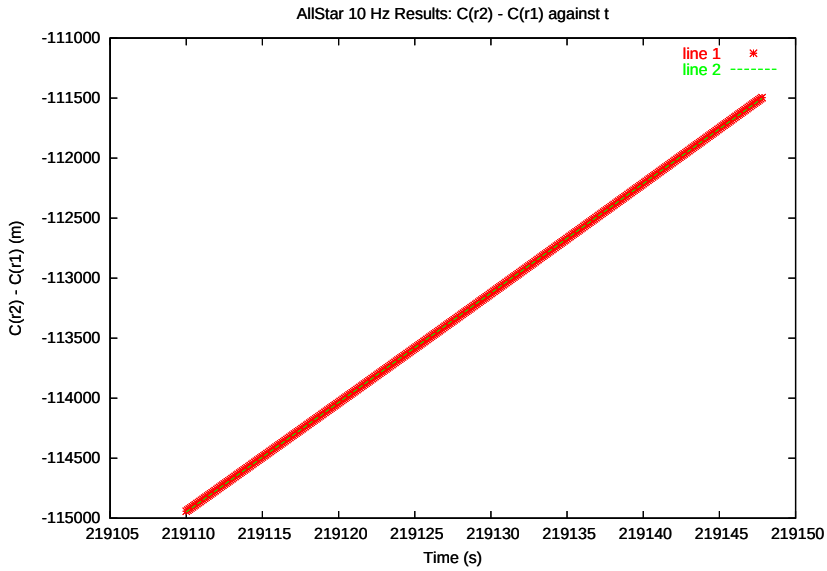

Figure B.14: 10 Hz AllStar raw data: $C_{r_2} - C_{r_1}$ against t — not flat as expected

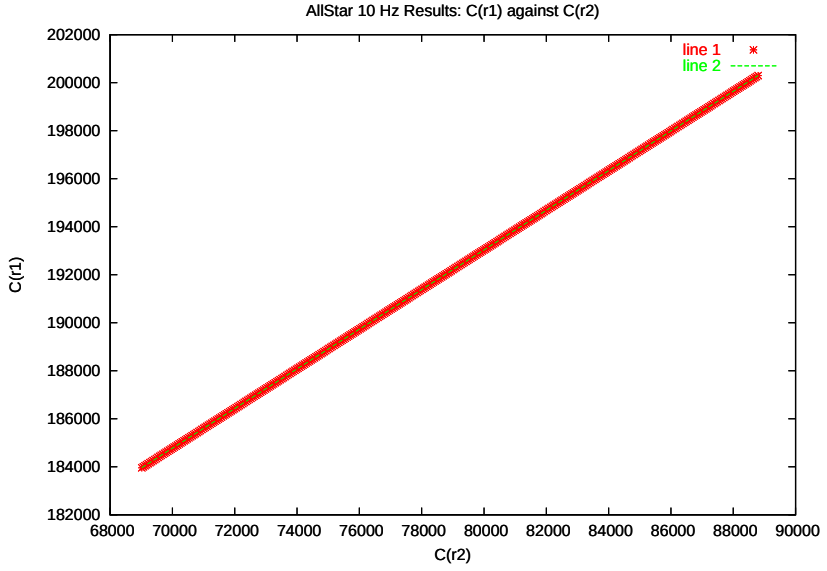

Figure B.15: 10 Hz AllStar raw data: C_{r_1} against C_{r_2} — gradient is not 1 as expected

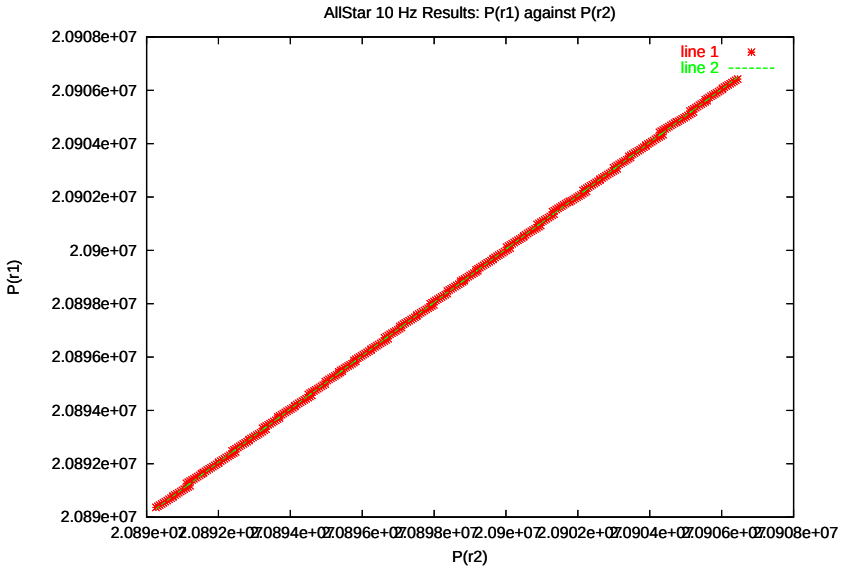

Figure B.16: 10 Hz AllStar raw data: P_{r_1} against P_{r_2} — gradient is 1

$$C_{r_1} = 0.825814 \times C_{r_2} + 126967.918364$$

Surely the phase information should be in lock step?

4. $\boldsymbol{P_{r_1}}$ against $\boldsymbol{P_{r_2}}$ at 10 Hz

   Graph B.16 is a straight line of gradient 1. A linear fit to the data gives:

$$P_{r_1} = 0.999622 \times P_{r_2} + 7946.303658$$

5. $\boldsymbol{C_t}$ against $\boldsymbol{P_t}$ for $\boldsymbol{r_1}$ at 10 Hz

   Graph B.17 is a straight line of gradient 1. A linear fit to the data gives:

$$C_t = 1.017866 \times P_t - 21079630.384684$$

The implication is that unit r_1 is behaving.

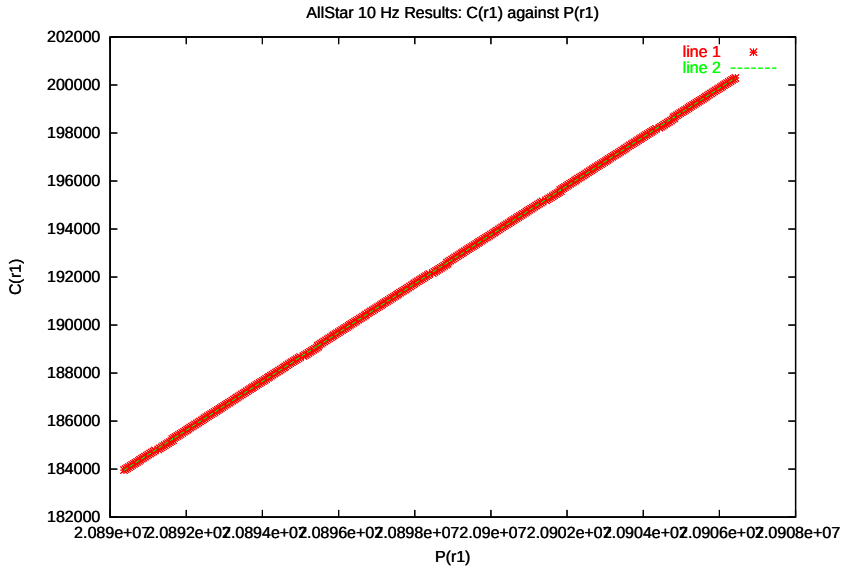

Figure B.17: 10 Hz AllStar raw data: C_t against P_t for r_1 — gradient is 1

6. $\boldsymbol{C_t}$ against $\boldsymbol{P_t}$ for $\boldsymbol{r_2}$ at 10 Hz

   Graph B.18 is a straight line. A linear fit to the data gives:

   $$C_t = 1.232109 \times P_t - 25670147.959454$$

   Why is the gradient not 1? The implication is that unit r_2 is misbehaving.

7. $\boldsymbol{P_{t_2} - P_{t_1}}$ against $\boldsymbol{t}$ for $\boldsymbol{r_1}$ at 10 Hz

   Graph B.19 is a straight line roughly constant first order difference of P. There are multiple blips which occur every tenth point, exactly and suspiciously **on** the second. A linear fit to the data gives:

   $$P_{t_2} - P_{t_1} = -0.000054 \times t + 54.287045$$

8. $\boldsymbol{P_{t_2} - P_{t_1}}$ against $\boldsymbol{t}$ for $\boldsymbol{r_2}$ at 10 Hz

   Graph B.20 is a straight line showing roughly constant first order difference of P. There are multiple downwards blips which occur every tenth point, exactly **on** the second. A linear fit to the data gives:

   $$P_{t_2} - P_{t_1} = 0.007009 \times t - 1493.076470$$

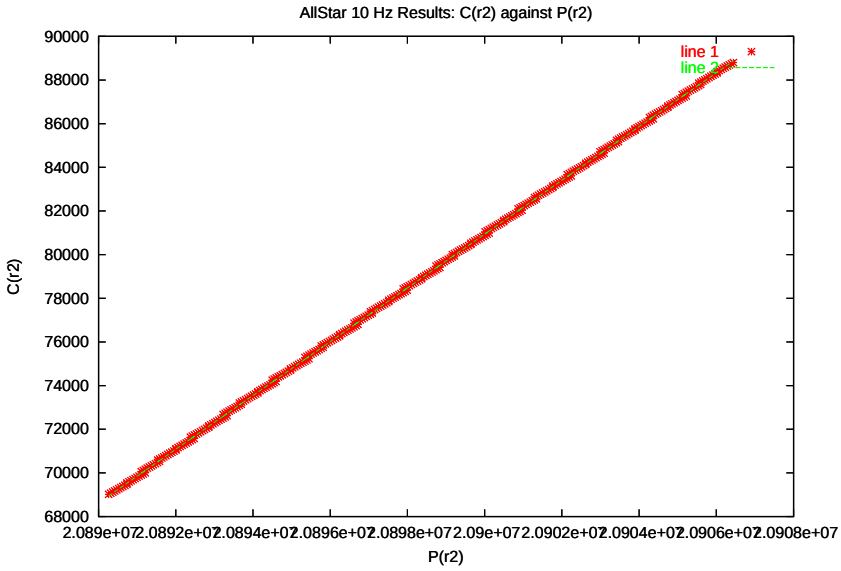

Figure B.18: 10 Hz AllStar raw data: C_t against P_t for r_2 — gradient not 1 as expected

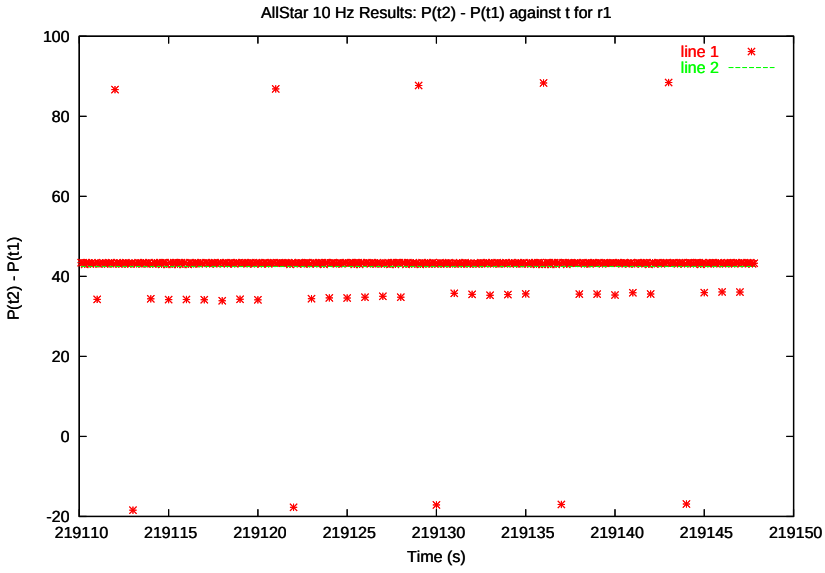

Figure B.19: 10 Hz AllStar raw data: $P_{t_2} - P_{t_1}$ against t for r_1 — flat with blips

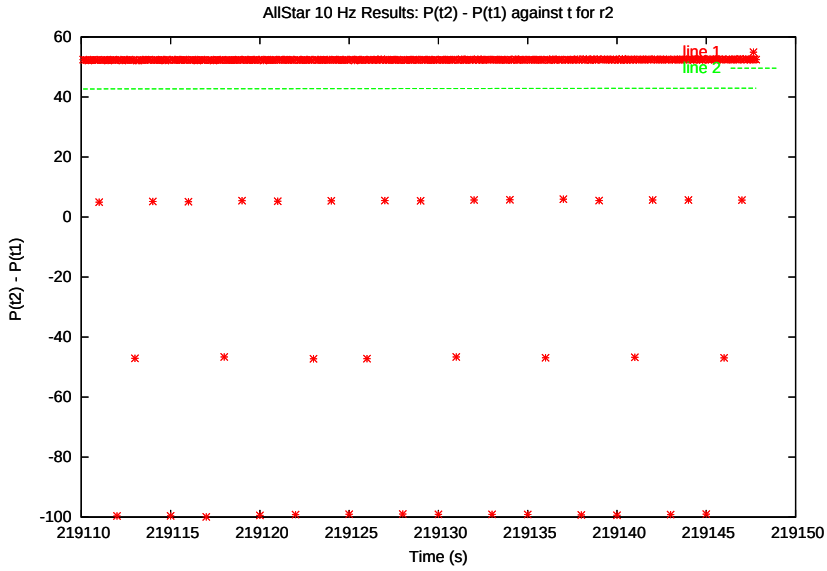

Figure B.20: 10 Hz AllStar raw data: $P_{t_2} - P_{t_1}$ against t for r_2 — flat with blips

9. $\boldsymbol{C_{t_2} - C_{t_1}}$ against $\boldsymbol{t}$ for $\boldsymbol{r_1}$ at 10 Hz

   Graph B.21 serpentines. There are multiple upwards blips which occur every tenth point, exactly **on** the second. A linear fit to the data gives:

$$C_{t_2} - C_{t_1} = 0.000443 \times t - 53.858549$$

10. $\boldsymbol{C_{t_2} - C_{t_1}}$ against $\boldsymbol{t}$ for $\boldsymbol{r2}$ at 10 Hz

    Graph B.22 does not serpentine. There are multiple upwards blips which occur every tenth point, exactly and suspiciously on the second. A linear fit to the data gives:

$$C_{t_2} - C_{t_1} = 0.005279 \times t - 1104.547711$$

11. $\boldsymbol{(P_{t_2} - P_{t_1}) - (C_{t_2} - C_{t_1})}$ against $\boldsymbol{t}$ for $\boldsymbol{r_1}$ at 10 Hz

    Graph B.23 is approximately a flat straight line (roughly constant value of zero) as expected. There are multiple blips which occur every tenth point, exactly **on** the second. A linear fit to the data gives:

$$(P_{t_2} - P_{t_1}) - (C_{t_2} - C_{t_1}) = -0.000497 \times t + 108.145593$$

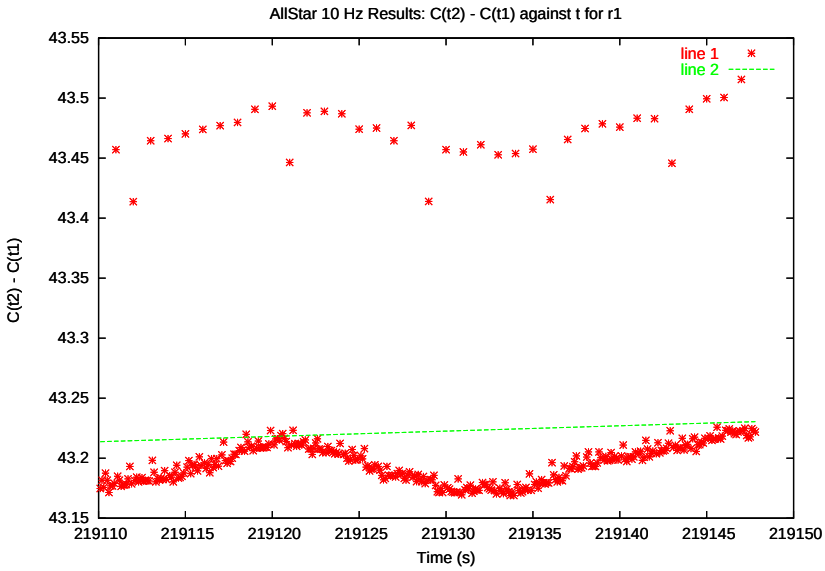

Figure B.21: 10 Hz AllStar raw data: $C_{t_2} - C_{t_1}$ against t for r_1 — serpentine with blips

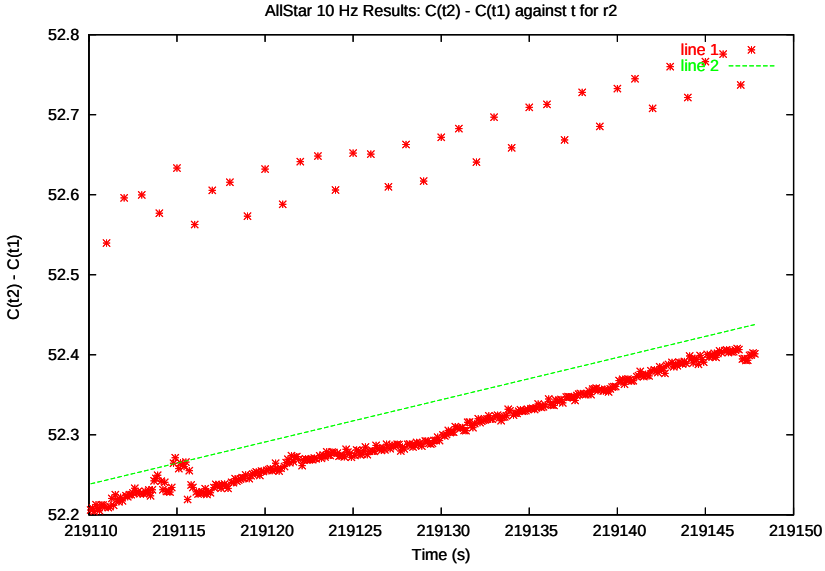

Figure B.22: 10 Hz AllStar raw data: $C_{t_2} - C_{t_1}$ against t for r_2 — non-serpentine with blips

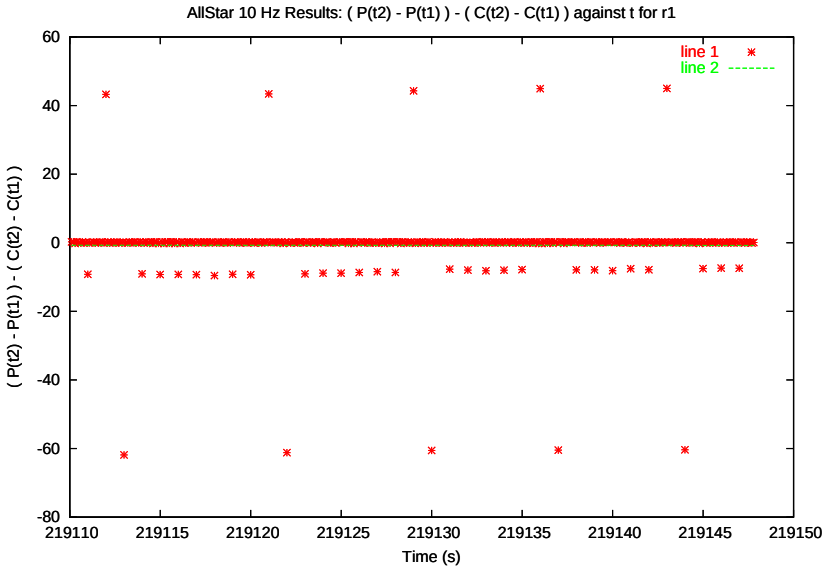

Figure B.23: 10 Hz AllStar raw data: $(P_{t_2} - P_{t_1}) - (C_{t_2} - C_{t_1})$ against t for r_1 — flat roughly zero

12. $\boldsymbol{(P_{t_2} - P_{t_1}) - (C_{t_2} - C_{t_1})}$ against $\boldsymbol{t}$ for $\boldsymbol{r_2}$ at 10 Hz

    Graph B.24 is approximately a flat straight line (roughly constant value of zero) as expected. There are multiple downwards blips which occur every tenth point, exactly and suspiciously **on** the second. A linear fit to the data gives:

$$(P_{t_2} - P_{t_1}) - (C_{t_2} - C_{t_1}) = 0.001730 \times t - 388.528759$$

B.3.2 Conclusion

P from r_2 is less well behaved than P from r_1. C from r_2 is changing at a different rate to C from r_1, whereas they should be be changing at the same rate. The implication is that phase measurements from r_2 are completely wrong - certainly enough to throw out the differential GPS calculations. In short the lower quality P from r_2 and the wrong C from r_2 may be symptoms of a problem with the unit. The data also shows bad results coming on the second! The inbetween results do not show any of these large blips. It is ironic that we have been working with the “on-the-second” figures for our positional calculations. In short, something particular seems to be wrong with r_2, and something in general seems to be wrong with r_1 and r_2.

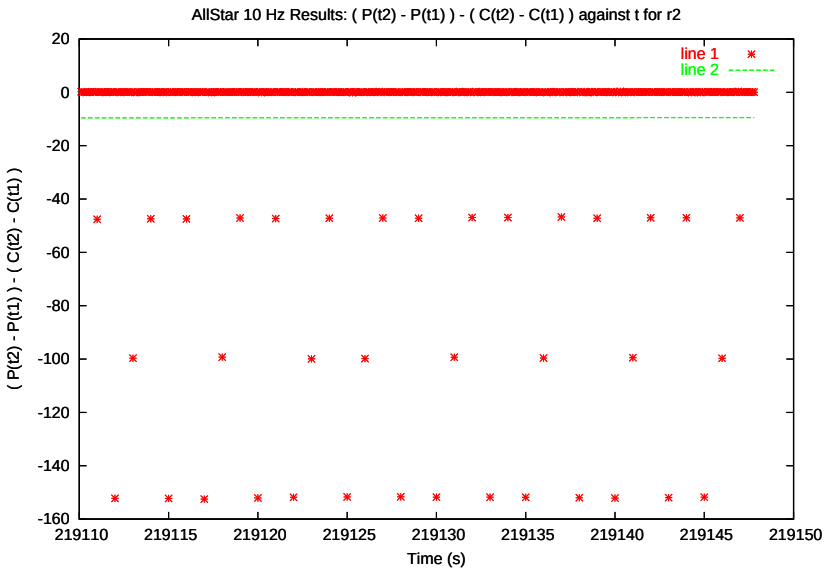

Figure B.24: 10 Hz AllStar raw data: $(P_{t_2} - P_{t_1}) - (C_{t_2} - C_{t_1})$ against t for r_2 — flat roughly zero

B.4 Investigating the Asymmetry

Suspecting that one receiver was worse that the other, their rôles were reversed in in the differential GPS calculation, on the off chance that this would improve the situation. Figure B.25 shows the effect, the level accuracy is unfortunately symmetric and there is **no** benefit in swapping the rôles of the receivers. The green track shows the smoothed differential GPS positions when the suspect receiver is used as reference, and the red track shows the corresponding positions when the non-suspect receiver is the reference.

B.5 Accuracy Plots

Figure B.26 gives a comparison between the standalone accuracy of the two receivers and the result from smoothed and non-smooth differential GPS. It is observable that differential operation gives no benefits in tightening the clustering, though the differential result is closer to the true position.

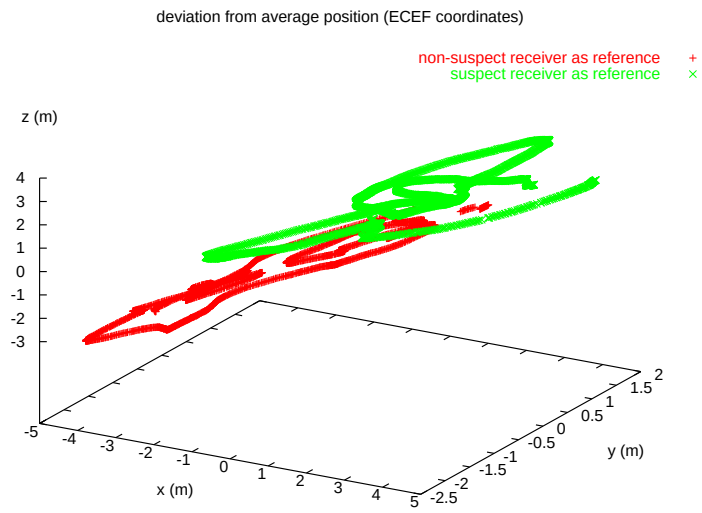

Figure B.25: Reversing mobile and reference rôles

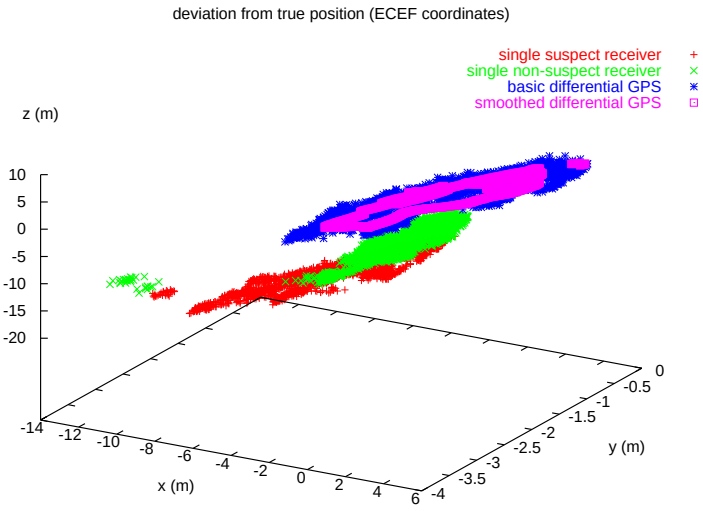

Figure B.26: Accuracy plots

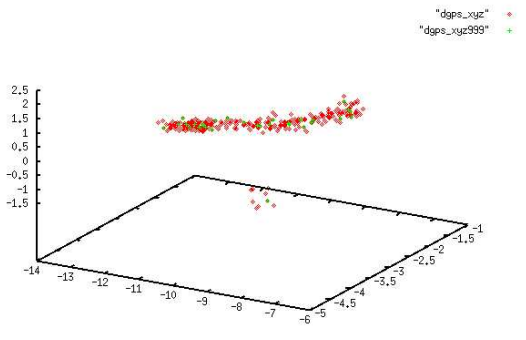

Figure B.27: DIY DGPS - few seconds of data at 10 Hz

Point Colour Allocated for Graph B.26	
Point Colour	DIY GPS Calculation
red	single suspect receiver
green	single non-suspect receiver
blue	basic differential GPS
cyan	smoothed differential GPS

B.6 Identifying the Culprit

The manner in which the determination of carrier phase goes wrong is strongly time dependent - witness the regular perturbing blips in the graphs and the difference between the on-second and off-second results above. The on-second phase and pseudorange appear particularly noisy. Perhaps something could still be salvaged from the equipment by selectively ignoring certain data on a time-basis? The following experiment was the final attempt. A succession of different position fixes were taken (both smoothed and unsmoothed) and plotted. The on-second results are coloured in green and the off-second results coloured in red.

The plots in Figures B.27 and B.28 show that the on-second results are well spread through the off-second results. Perhaps the noisy phase and pseudorange on the second cancel out somehow? Anyhow, the major culprit would thus appear to the out of control carrier phase information from one or both of the receivers. This is supported by the way the point locus zooms off!

The phase smoothing indicates the potential improvement in positional accuracy *i.e.* a cross sec-

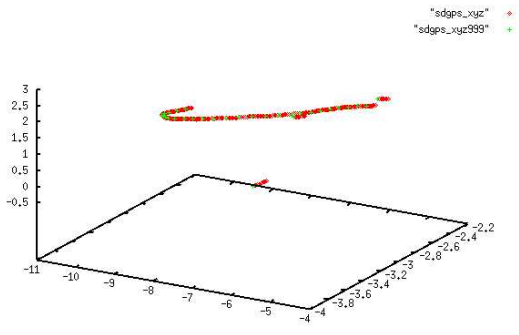

Figure B.28: DIY DGPS with phase smoothing - few seconds of data at 10 Hz

tion of the locus here is much more tightly bundled. If only the “zoom off” did not exist - attributable to bad carrier phase information from one or both of the boards.

The comparison is not quite as clear as it might be. In Figure B.28 the on-second data is used to smooth the off-second data (rather than being ignored totally as potentially suspect). However, the first plot indicates that the on-second data is not wildly out - so the red points in the second plot are not much different from how they would be in an off-second data only solution.

Appendix C

Date Conventions

C.1 Introduction

The RINEX file format incorporates full date information, whereas the GPS unit merely returns the GPS second, which is a number between 0 and 604800. This identifies a particular second within a GPS week, but does not uniquely identify a date.

So the mapping from the GPS hardware supplied data and a RINEX file is not well defined, and could be resolved say by using the system clock on the host computer. This resolution depends on knowing where the exact boundaries of the GPS weeks are within the calendar - extra information which would have to be researched and/or derived and so could be a source of error. However, the LAMBDA code itself extracts and indeed only uses the GPS second derived from the date and time of the observation, so in a sense the date written to the RINEX file does not have to be correct, but only needs to reveal the correct GPS second.

One zero point of the GPS second is implicitly known, because it can be derived from the combination of the sample GPS data and the existing LAMBDA software. Let us consider one sample reference observation taken at 04:26:00.0 on 23/10/1996. We can convert this into a real number, whose whole part is the Julian day number and whose fractional part gives the elapsed fraction of the day, using the routine `date_to_julian` supplied as part of the LAMBDA software:

```
double date_to_julian(int y, int m, int d, int H, int M, double S)
// software already supplied - date in format HH:MM:SS.S  dd/mm/yyyy
{
        if ( m <= 2 )
        {
                   y = y-1;
                   m = m+12;
        }
        jd = floor(365.25*y) + floor(30.6001*(m+1))
           + d + (H+M/60.0+S/3600.0)/24 + 1720981.5;
}
```

The Julian day is just a standard way of numbering days as successive integers to simplify calculations by removing considerations of year, month and date. In order to create (instead of just interpret) RINEX files it proved necessary to write an inverse operation, which turns a real-valued Julian day into a date:-

```
void julian_to_date
(
        double julian,
        int *y, int *m, int *d, int *H, int *M, double *S
)
{
        double h;
        int jd, l, n, i, j;

        jd =  (int) floor( julian + 0.5 ); // Julian days start at mid-day
        h = 24.0 * ( (julian + 0.5) - jd );

        (*H) =  (int)   floor(h);
        (*M)  = (int)   floor( 60.0 * ( h - (*H) ) );
        (*S)  = 3600.0 * ( h - (*H) - (*M) / 60.0 );

         l = jd + 68569;
         n = ( 4 * l ) / 146097;
         l = l - ( 146097 * n + 3 ) / 4;
         i = ( 4000 * ( l + 1 ) ) / 1461001;
         l = l - ( 1461 * i ) / 4 + 31;
         j = ( 80 * l ) / 2447;
         *d = l - ( 2447 * j ) / 80;
         l = j / 11;
         *m = j + 2 - ( 12 * l );
         *y = 100 * ( n - 49 ) + i + l;
}
```

The method of computing the inverse had to be hunted-down on the Internet. The extraction of a date from the Julian day is neither a trivial nor obvious calculation as can be seen from the code, and this is why it is presented here! Using both these utilities, the code was then written to produce a plausible date from a GPS time.

```
void gpstime_to_date
(
        double sec_of_week,
        int *y, int *m, int *d, int * H, int * M, double * S
)
{
        double julian_ref;
        double julian;

        julian_ref     = date_to_julian(1996,10,23, 4 , 26, 0.0);
        julian_to_gpstime( &week_ref, &sec_of_week_ref, julian_ref);
        julian = julian_ref - (sec_of_week_ref/86400.0)  + (sec_of_week/86400.0);
        julian_to_date( julian, y, m, d, H, M, S);
}
```

A support function `julian_to_gpstime` is required which turns a real-valued Julian date into a GPS time. This is supplied as part of the LAMBDA code, and is presented as again it is far from trivial.

```
void julian_to_gpstime(double julian, int * week, double * sec_of_week)
// software already supplied
{
        int a;
        int b;
        int c;
        double d;
        int e;
        int f;
        int day_of_week;

        a = (int) floor(julian+.5);
        b = a+1537;
        c = (int) floor((b-122.1)/365.25);
        e = (int) floor(365.25*c);
        f = (int) floor((b-e)/30.6001);
        d = b-e-floor(30.6001*f)+rem(julian+.5,1.0);
        day_of_week =  ( (int) floor(julian+.5))  % 7;
        (*week) = (int) floor((julian-2444244.5)/7);
        //% We add +1 as the GPS week starts at Saturday midnight
        (*sec_of_week) = (rem(d,1.0)+day_of_week+1)*86400;
}
```

The necessary derivation of `gpstime_to_date` to produce RINEX files is subtle, and is shown in Figure C.1. The broad arrow shows the transformation of data that we require. The direct way to do this would require the non-existent function `gpstime_to_julian`, which is represented by the dotted line. In the absence of knowing where the zero point is in the GPS week to directly write this function, or equivalently in the attempt to avoid error by not writing new code, such a function was circumvented, by using actual data and existing functions to establish these zero points.

Now the reference date we used was essentially arbitrary, However, the particular values used were retained because they allowed the testing of `julian_to_date` and `data_to_julian` as

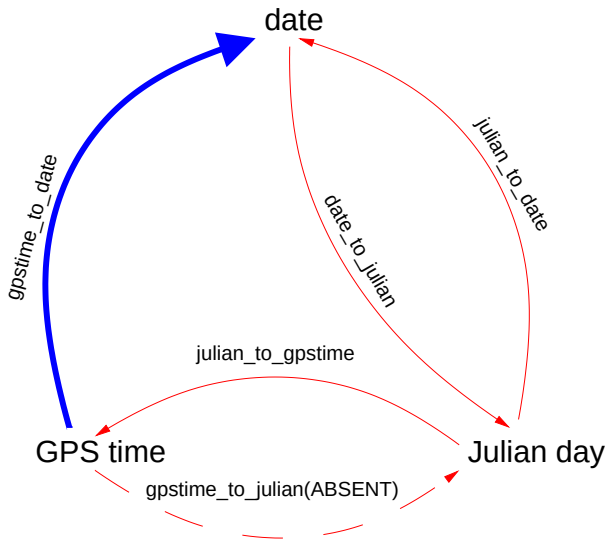

Figure C.1: Date conversion functions

inverse operations *i.e.* the RINEX files written proved identical to the ones read when the software was specially set-up for testing. For capturing live RINEX files from hardware, this reference date should clearly be taken from the system clock of the host computer. This was not done in the current context, because it is immaterial for the nature of the GPS positional solutions obtained.

Appendix D

Utility Software: the dev2soc Program

D.1 dev2soc Documentation

dev2soc is a utility that was originally developed for real-time GPS processing software to transparently connect to GPS units attached to remote machines on the network. dev2soc runs on the remote machine and connects to the local machine via a socket.

More generally dev2soc connects an RS232 device physically attached to a remote machine to a UNIX socket, allowing it to be used just like a local RS232 device. Existing code remains largely unmodified: just open the socket instead of the local RS232 port - the rest of the existing code will work as before. Why buy a new RS232 port when you can borrow one on another machine? dev2soc provides up to 20 kbytes of buffering, which will overcome many of the problems of data loss on RS232 connections.

```
   USAGE  :  dev2soc -P <device> -S <socket no> -B <baud rate>
DEFAULT  :  dev2soc -P /dev/ttyS0 -S 2104 -B 9600
```

D.2 dev2soc Code

The code for the core function transfer within dev2soc is given below. This handles the unidirectional transfer of data between an GPS device and a socket, and *vice versa*.

```
#define NO_EVENT 0
#define READ_EVENT 1
#define WRITE_EVENT 2
#define READ_AND_WRITE_EVENT ( READ_EVENT | WRITE_EVENT )
#define BUFSIZE 20000

void transfer(int fd_in, int fd_out)
{
        int n;                      // no of bytes read or written
        int retval;                 // bits indicate whether a read or write can take place
        char buf[BUFSIZE + 10];     // communications buffer
        char * read_buf_ptr = buf;  //  set reading pointer to start of buffer
        char * write_buf_ptr = buf; //  set writing pointer to start of buffer
        int n_remaining = BUFSIZE;  //  empty spaces in buffer
        int n_waiting   = 0;        //  bytes in buffer
        for(;;)
        {
                // fcntl(soc, F_SETFL, O_NONBLOCK) ???????
                // may return fewer bytes than expected = that's good!

                if ( n_waiting == 0 )
                {
                        retval = wait_on_read(fd_in);        // no data in buffer -> wait for read event
                }
                else if ( n_remaining == 0 )
                {
                        retval = wait_on_write(fd_out);      // no space in buffer -> wait for write event
                }
                else
                {
                        retval = wait_on_read_and_write( fd_in, fd_out);    // wait for read or write event
                }

                if ( retval & WRITE_EVENT ) // do write first to move data through quickly
                {
                        if ( n_waiting > 0 )
                        {
                                n = write(fd_out,write_buf_ptr,n_waiting); // write up to n_waiting bytes
                                if ( n > 0 )                              // n = no of bytes written
                                {
                                        write_buf_ptr += n; n_waiting     -= n;
                                        if ( n_waiting == 0 ) // all written out
                                        {
                                                read_buf_ptr = buf; write_buf_ptr = buf; n_remaining = BUFSIZE;
                                        }

                                }
                        }
                }

                if ( retval & READ_EVENT ) // then do read
                {
                        if ( n_remaining > 0 )
                        {
                                n = read(fd_in,read_buf_ptr,n_remaining);       // read up to n_remaining bytes
                                if ( n > 0)                                     // n = no of bytes read
                                {
                                        n_remaining -= n; read_buf_ptr += n; n_waiting += n;
                                }
                        }
                }
        }
}
```

The functions `wait_on_read`, `wait_on_write` and `wait_on_read_and_write` are self-explanatory in function. They were originally written in terms of the UNIX select system call, which determines whether a number of communication channels are ready to communicate. However, select was found NOT to work, so these were written instead in terms of the UNIX poll system call which has similar functionality. The read and write are the basic UNIX system calls for lowest level communication.

Appendix E

Inverting a Monotonic Function

```
(* FUNCTION INVERSION ALGORITHM IN ML
   ---------------------------------
   Finds inverse of a strictly monotonic real function within given range.

   Arguments      Description
   (x0,x1)        range - order of range endpoints is not important
   f              function to be inverted
   target         argument to inverse function
*)
fun monotonic_range_inverse x0 x1 (f:real->real) target =
let
       fun fabs x    = if ( x > 0.0 ) then x else ~x;
       val wee       = 1.0 / Math.pow(real Real.radix, real Real.precision);
       fun equal x y = ( fabs(x - y ) < wee );
                       (* to within floating-point representation     *)
       nonfix in_range;
       fun in_range (x:real, (x0,x1) ) = ( (x0 < x) andalso (x < x1) )
                                         orelse
                                         ( (x1 < x) andalso (x < x0) );
       infix in_range; (* in_range is defined as nonfix but used infix *)
in
       if ( equal  x0  x1  )
       then
               x0
       else
               let
                       val y0  = f x0;
                       val x   = (x0 + x1) / 2.0;
                       val y   = f x;
               in
                       if ( target in_range (y0,y) )
                       then monotonic_range_inverse x0 x  f target
                       else monotonic_range_inverse x  x1 f target
               end
end;

(* for full inverse function extend to full real range *)
val monotonic_inverse = monotonic_range_inverse (~Real.maxFinite) Real.maxFinite;

(* invert erf function using monotonic inversion operator *)
val erf_inverse = monotonic_inverse erf;
```

Appendix F

Optimising Region Filtering

The traditional code to work out an image of region average intensities (AV) is of the following form. Let the region width be $2n_x + 1$; the region height be $2n_y + 1$; the image width be w and the image height be h.

```
count = (2 * nx + 1) * (2 * ny + 1);
for( y = ny; y < h - ny; ++ y)
{
        for(x = nx; x < w - nx; ++ x)
        {
                sum = 0;
                for ( i = -ny ; i <= ny ; ++ i )
                {
                        for(j = -nx ;  j <= nx ; ++ j )
                        {
                                sum = sum + I[x+j,y+i];
                        }
                }
                AV[x,y] = sum / count;
        }
}
```

Note the computational complexity is high at $O(whn_xn_y)$, and many additions are being repeated unnecessarily. The first optimisation is to separate the filter: first summing the values horizontally and then summing these values vertically.

```
        for( y = 0; y < h; ++ y)
        {
                for(x = nx; x < w - nx; ++ x)
                {
                        for(sum = 0, j = -nx ;  j <= nx ; ++ j )
                        {
                                sum = sum + I[x+j,y]
                        }
                        SUM_H[x,y] = sum;
                }
        }
        count = (2 * nx + 1) * (2 * ny + 1);
        for( y = ny; y < h - ny; ++ y)
        {
                for(x = nx; x < w - nx; ++ x)
                {
                        sum = 0;
                        for ( i = -ny ; i <= ny ; ++ i )
                        {
                                sum = sum + SUM_H[x,y+i]
                        }
                        AV[x,y] = sum / count;
                }
        }
```

The computational complexity is reduced by a multiplicative term to become $O(wh(n_x + n_y))$. As the now linear filter is shifted by one pixel, the sum is increased by the value of the pixel covered and decreased by the value of the pixel uncovered. Thus by using a further "sliding window" optimised approach, only 2 additions are required per pixel. To make this change possible in the vertical direction, the outer loops of the second pass have to be swapped.

```
for( y = 0; y < h; ++ y)
{
        x   = nx;
        sum = 0;
        for(j = -nx ;  j <= nx ; ++ j )
        {
                sum = sum + I[x+j,y];       // x coordinate goes from 0 to 2 * nx
        }
        SUM_H[x,y] = sum;

        for(x = nx + 1; x < w - nx; ++ x)
        {
                sum = sum + I[x+nx, y] - I[x-nx-1, y];
                SUM_H[x,y] = sum;
        }
}
count = (2 * nx + 1) * (2 * ny + 1);
for(x = nx; x < w - nx; ++ x)
{
        y   = ny;
        sum = 0;
        for ( i = -ny ; i <= ny ; ++ i )
        {
                sum = sum + SUM_H[x,y+i];   // y coordinate goes from 0 to 2 * ny
        }
        AV[x,y] = sum / count;

        for( y = ny + 1; y < h - ny; ++ y)
        {
                sum = sum + SUM_H[x,y+ny] - SUM_H[x,y-ny-1];
                AV[x,y] = sum / count;
        }
}
```

The computational complexity is now $O(hw)$, reduced again by a multiplicative term. The next manipulation is to remove the serial dependency on the variable `sum`. This permits the loops being broken up further, and allows the swapping of the outermost loops in the 4^{th} top level loop below. By swapping these loops we step through the 2D arrays horizontally instead of vertically. In memory terms we move from one item to the next in memory, instead of the one w elements away, allowing more efficient and consistent addressing.

```
for( y = 0; y < h; ++ y)       // loop 1
{
       x   = nx;
       SUM_H[x,y] = 0;
       for(j = -nx ;  j <= nx ; ++ j )
       {
              SUM_H[x,y] = SUM_H[x,y] + I[x+j,y]; // x coord goes from 0 to 2 * nx
       }
}

for( y = 0; y < h; ++ y)       // loop 2
{
       for(x = nx + 1; x < w - nx; ++ x)
       {
              SUM_H[x,y] = SUM_H[x-1,y]  + I[x + nx, y] - I[x - nx - 1, y];
       }
}

for(x = nx; x < w - nx; ++ x)  // loop 3
{
       y = ny;
       for ( i = -ny ; i <= ny ; ++ i )
       {
              AV[x,y] = AV[x,y] + SUM_H[x,y + i]; // y coord goes from 0 to 2 * ny
       }
}

for(x = nx; x < w - nx; ++ x)  // loop 4
{
       for( y = ny + 1; y < h - ny; ++ y)
       {
              AV[x,y] = AV[x,y - 1] + SUM_H[x,y + ny] - SUM_H[x,y - ny - 1];
       }
}

count = (2 * nx + 1) * (2 * ny + 1);
for(x = nx; x < w - nx; ++ x)  // loop 5
{
       for( y = ny ; y < h - ny; ++ y)
       {
              AV[x,y] = AV[x,y] / count;
       }
}
```

The 2^{nd} and 4^{th} top level loops are the computationally intensive loops. Loops 1 and 3 are small initialisation loops, and loop 5 is a small termination loop. Loops 2 and 4 are now essentially the same loop (albeit parameterised slightly differently). We aim for a common representation, for simultaneous optimisation. Thinking in terms of four linear memory indices in the code line:

```
SUM_H[x,y] = SUM_H[x-1,y] +  I[x + nx, y] - I[x - nx - 1, y];
```

it can be observed these would stay at constant relative offsets from each other: namely 0, -1, n_x, and $-n_x - 1$. Thinking in terms of four linear memory indices in the code line:

```
AV[x,y] = AV[x,y - 1] + SUM_H[x,y + ny] - SUM_H[x,y - ny - 1]
```

these would similarly stay at constant relative offsets from each other: namely 0, $-w$, $n_y w$, and $-n_y w - w$. A common routine must therefore be parameterised by two positive offsets, *offset1* and *offset*. These generate resultant offsets 0, *-offset1*, *offset2* and *-offset2-offset1*. Two of the terms are the same source array, and two of the terms are the same destination array. By creating different views of the source and destination arrays *i.e.* by using different pointer variables offset by the required amounts, it is possible to use the same index variable for all terms in the expression. By working through a length of memory containing the 2D sub-arrays of interest, instead of just the 2D sub-arrays themselves, the processing takes on the following attributes:

1. The computational kernel becomes a single instead of a double loop
2. Some unnecessary computation will be done but the proportionate loop overheads are lower.
3. Only a single array element (instead of an array of elements) is required to be initialised, in top level loop 1. This is not true for top level loop 3, however, due to the difference in direction.

Let us call the routine calling the computational kernel `slur`. The code becomes:

```
        x   = nx;
        y   = 0;
        SUM_H[x,y] = 0;
        for(j = -nx ;  j <= nx ; ++ j )
        {
                SUM_H[x,y] = SUM_H[x,y] + I[x + j,y];   // x goes from 0 to 2*nx

        }

        slur( SUM_H, I, w * h , nx + 1, nx );           // horizontal slur

        for(x = nx; x < w - nx; ++ x)
        {
                y = ny;
                for ( i = -ny ; i <= ny ; ++ i )
                {
                        AV[x,y] = AV[x,y] + SUM_H[x,y + i];
                }
        }

        slur(SUM_H, I, w * h - (2 * ny), ny * w + w,  ny * w);// vertical slur

        count = (2 * nx + 1) * (2 * ny + 1);

        for(x = nx; x < w - nx; ++ x)
        {
                for( y = ny ; y < h - ny; ++ y)
                {
                        AV[x,y] = AV[x,y] / count;
                }
        }
```

where the finalised `slur` routine is given below:

```
void slur( SLUR_TYPE * d, SLUR_TYPE * s, int npixels, int offset1, int offset2)
{
        int count;
        SLUR_TYPE * d2, * d3, *s2;

        count = npixels - (offset1 + offset2);
        d2 = d + offset2; d3 = d + offset1;
        s2 = s + offset2 + offset1;

        while(count--)
        {
                d2[count] = d3[count] + s[count] - s2[count];
        }
}
```

In essence the whole process of image averaging (which takes the majority of the processing time) has been reduced to its essential kernel which is a single line of code. Can we optimise this single line of code further? The `slur` routine was consequently written in terms of the Pentium MMX instruction set. Notice the address arguments are now committed to a particular type, as we are nearer the hardware.

For the vertical slur, the MMX code had identical performance to the 'C' code. For the horizontal slur, the memory addressing pattern made the code unsuitable for a translation to the MMX instruction set. The problem was the horizontal averaging: addresses `d2` and `d3` are only one 32-bit word apart. MMX optimisation works 64-bits at a time - so the reading from `d3` and writing to `d2` interfere with one another.

```
void slur_mmx(int * d, int * s, int npixels, int offset1, int offset2)
{
    int count;
    int * d2, * d3, *s2;
    mmx_t ma,mb,mc;

    count = npixels - (offset1 + offset2);

    count += 2;
    d     -= 2;
    s     -= 2;

    d2 = d + offset2; d3 = d + offset1;
    s2 = s + offset2 + offset1;

    if(!mmx_ok()) exit(-1);

    if ( count & 1 ) count--;
    while(count-=2)
    {
        movq_m2r(d3[count], MM0);  //load b3[] with a quad word
        paddd_m2r(s[count], MM0);  //add a[] using saturated
        psubd_m2r(s2[count], MM0); //subtract a2
        movq_r2m(MM0,d2[count]) ;  //move quad word result to b2
    }

    emms(); /*leave mmx, go back to FP mode*/
}
```

The conclusion is that using MMX instructions has no benefit whatsoever in this circumstance, and it is pleasing for portability to be able to keep the source in the final optimised ‘C’ form, knowing that this code is the fastest.

Appendix G

Visual Positioning System API

```
//-------------------------------------------------------
//File vps_api.h
//-------------------------------------------------------

#include <stdio.h>
#include <mymalloc.h>
#include <string.h>
#include <fstream.h>
#include <math.h>

#include "hvci.h"
#include "X_simple.h"
#include "region.h"
#include "target.h"
#include "graph.h"
#include "matrix.h"
#include "verify.h"
#include "debug.h"

/*

VPS - A Robust Visual Positioning System
========================================
========================================

1. Introduction
===============
The Visual Positioning System (VPS) allows a live video stream to be used as a position
determination device. Provided at least one specially-designed VPS target is in view then
the system can determine the position of the camera in world coordinates.

This document explains the capability and functioning of the VPS. In particular, it
documents the Application Programmer Interface (API) for the VPS and the supplied exemplar
VPS application, which had been designed to place the VPS capability within a useful
flexible context.

2. Hardware Requirements
========================
The VPS requires the following hardware components:

   (1) a video camera (mono or colour)
   (2) a video-grabber board based on the Brooktree chip (e.g. Hauppauge range of TV-cards)
   (3) a PC running Linux, with video-grabbing capable kernel
   (4) laser-printer (mono or colour) to print out VPS targets, which are supplied as
       PostScript files
```

3. Software Requirements

The VPS requires the following software components

(1) Linux, with kernel capable of video-grabbing
(2) X-Windows
(3) VPS library (supplied)
(4) X-Simple library (a simple interface to X-Windows - supplied)
(5) HVCI library (a simple video-grabbing interface - supplied)

Note that both (4) and (5), which are software modules developed in-house at Essex University, are supplied with the VPS. While VPS does not directly call X-Simple and HVCI, it does use the data structures which they define. The source of the exemplar VPS application (supplied) illustrates the use of the HVCI for image input and X-Simple for image output.

The VPS code itself has no particular machine dependencies. It should be portable across UNIX systems with little/no effort, and in theory could run within a Windows environment.

4. VPS Application Capability

The supplied VPS application may take its input from a single PNM file; a sequence of PNM files or a live video feed. The file and video-grabs must have 256 gray levels. The output may include sequences of PNM files derived from any stage in the image processing pipeline; calculated calibration coefficients; and position and orientation fixes.

5. VPS Application Documentation

The command line options are:

```
-help                                   print this help
-w<default val = 10>                    set half filter width and height
-v<default val = 4>                     set gray variance threshold
-p<default is off>                      calculate position
-r<default is off>                      use root X window for image display
-d<default is off>                      print debugging info
-D<default is off>                      print debugging info on camera position calculation
-F<default is off>                      full processing
                                        (NULL processing causes simple video display)
-O<default is off>                      measure performance
-P<default val = 20>                    pixel limit at which to ignore regions
-f<default val = 1.200000>              scale factor for video grabbing
-g<default val = 8>                     video grabbing depth
-i<default val = 100000>                no of iterations
-bn<base_name>                          base name for image files I/O
-o<default val = 1>                     Type of input
1 = use video
2 = use frames from hard disk
3 = use single frame from hard disk
-s<default val = 15>                    stages in image processing to display (OR values)
1 = source image
2 = 3-level gray image
4 = region image
8 = augmented image (source image + overlay-ed graphics)
-t<default val = 0>                     stages in image processing to save to file (OR values)
1 = source image                        <base_name>_src.pnm
2 = 3-level gray image                  <base_name>_gray.pnm
4 = region image                        <base_name>_region.pnm
8 = augmented image (source image + overlay-ed graphics)   <base_name>_aug.pnm
-labels<default val = 1>                show target IDs on augmented image
-cb<calibration file name>              name of file holding calibration coefficients
-TF<tagerget file name>                 name of file holding target details
                                        current examples are:

                                          2: calibration_rig.dat
                                          5: cube.dat
                                          6: ceiling.dat

-x<default val = 0>                     supply start number for image sequence
```

-y<default val = 0> supply finish number for image sequence
-c<default is off> calibrate camera
-U<default is off> find unidentified targets mode

6. VPS APS Capability

Each VPS target supplies 4 points of known position. These are the centres of the four white rectangles at the corners of each rectangular target. If a single target is found within the video image, then a mapping can be established between the known 3D positions of the target regions and their corresponding 2D locations within the video image. This mapping contains enough information to work out camera position.

The greater number of targets found in the scene the greater the accuracy obtained. If 6 or more non-coplanar points of correspondence are found, then the vital calibration information about the camera (e.g. focal length) can also be derived. If fewer than 4 points of correspondence are found then no positional information can be obtained. If the correspondence points are coplanar, or if there are between 4 and 5 non-coplanar correspondence points, then prior calibration coefficients pre-loaded from a data file are used.

Information about the targets' size, ID and location in world coordinates must be supplied by the user in a data file. This is called the "survey data".

With targets placed on fixed objects in the environment, the system reports on the position and orientation of the camera in world coordinates. With targets placed on a mobile object in front of a stationary camera, the system reports on the position and orientation of the object within the camera coordinate system. These transformations take the form of 4x4 matrices.

If a target is known to be misidentified by virtue of the fact that its ID should not have been seen, it is possible to reassign this target to a likely correct ID if a geometric mapping can be established without this target. The corrected target set can then be used to establish a more accurate mapping. Note that this capability within the API is an historical artifact from an earlier target ID encoding scheme which was more likely to fail. Note that with the present encoding scheme no identification failure has been observed. However, this functionality has been retained for insurance purposes.

7. VPS API Philosophy

A utility rountine "VPS_image_to_position()" has been provided which effectively turns a video still into a position fix. However, the API has been divided into 17 lower-level functions (below) for a number of reasons.

(1) Flexibility

The conversion of an image to a position fix is a pipelined operation with a number of sequenced components: image processing; subdivision into regions; target recognition; target identification; geometric numeric processing.

It may be that a user of the VPS does not want to follow through this pipeline completely, and the interface allows selective use of the pipeline components.

For example, the VPS turns the image into regions - and the programmer may wish to perform further region processing of their own.

(2) Pipeline Debugging

It is useful to be able to view intermediate images in the pipeline - these are invaluable to an understanding of operation of the VPS. For example looking at these images will instantly tell the user whether the input image is too dark for target recognition, or whether supplied algorithmic threshold values are incorrect.

Note that although the VPS is robust to threshold values it is still possible to supply inappropriate values.

The component pipeline processes bring these images to the level of the application programmer, so a choice is presented as to what to do with these images. Otherwise, this information would be hidden in the depth of a monolithic operation - and there would be no clues as to what was going wrong if the VPS did not provide an expected position fix.

The overall pipeline can fail is a number of ways:

* no targets found
* geometric mapping cannot be established
* geometric mapping cannot be split into interior and exterior matrices

However, each individual pipeline component can only fail in a single well-defined way. Hence the VPS allows the programmer to customise the response to failure.

(3) Performance

Many of the operations in the VPS are relatively computationally expensive. Performance may be optimised for any particular application, by using only the appropriate routines.

Note that is is perfectly possible to gather details about all regions within a image using a single function call. However, the VPS interface only collects enough information about regions sufficient for the next stage of processing. Indeed, only those regions identified as part of targets are processed beyond a certain stage.

The processing pipeline is like an inverted pyramid, where as much subsequent effort is saved by earlier filtering.

8. VPS API Data Structures

The following data structures are provided within the VPS API:

VPS_RegionTable

Holds details of regions within an image.

VPS_Graph

A graph structure holding a representation of the interconnectedness of regions within an image. Nodes are regions and arcs represent adjacency. These graphs are known as Region Adjacency Graphs (RAGs).

VPS_WorldTargetTable

Holds IDs of targets in the environment and their world coordinates. This information is typically loaded from a user supplied data file.

VPS_SceneTargetTable

Holds IDs of targets visible in the video grab, and their pixel coordinates in image space.

VPS_RecognisedTargetTable

The coalition of a VPS_WorldTargetTable and a VPS_SceneTargetTable i.e. combined details of targets both found in the 2D scene and known to be in the 3D environment.

VPS_Mapping

Geometric mapping derived from information in a VPS_RecognisedTargetTable structure.

VPS_CalibrationCoefficients

Calibration information about a camera. This is typically loaded from a user-supplied data file.

VPS_Camera

This holds the geometric mapping split into two component matrices: the geometric transform between object and camera coordinates (exterior orientation) and the projection matrix for the camera (interior orientation).

The following utility routines have been written to allocate and deallocate VPS data structures dynamically. There is only a single deallocate routine, as the structures have a common identifier field containing a different value for each structure type.

```
VPS_Graph                           * alloc_VPS_Graph  ();
VPS_WorldTargetTable                * alloc_VPS_WorldTargetTable  ();
VPS_CalibrationCoefficients         * alloc_VPS_CalibrationCoefficients  ();
VPS_RegionTable                     * alloc_VPS_RegionTable  ();
VPS_SceneTargetTable                * alloc_VPS_SceneTargetTable  ();
VPS_RecognisedTargetTable           * alloc_VPS_RecognisedTargetTable  ();
VPS_Mapping                         * alloc_VPS_Mapping  ();
VPS_Camera                          * alloc_VPS_Camera  ();

void free_VPS( void * ptr );

9. VPS API Documentation
========================
*/

/*
 * ######################
 * # SETTING THINGS UP #
 * ######################
 */

VPS_Graph * VPS_load_target_RAG(char * filename);
//-----------------------------------------------
/*
 *        Load details of key RAG from file.
 */
VPS_WorldTargetTable * VPS_load_world_targets(char * target_name);
//----------------------------------------------------------------
/*
 *        Load details of targets in world from file.
 */
VPS_CalibrationCoefficients * VPS_load_calibration(char * filename, int width, int height);
//-------------------------------------------------------------------------------------------
/*
 *        Load previously computed calibration coefficients of camera from file.
 *        Note that width and height of images being processed need to supplied
 *        in order to scale the coefficients correctly for use.
 */

/*
 * ####################
 * # IMAGE PROCESSING #
 * ####################
 */

Pic * VPS_local_equalise
(
        Pic * p,
        int half_filter_width, int half_filter_height,
        int standard_deviation_threshold
);
//-----------------------------------------------------
/*
 *      Create 3 gray-level image from 256 gray-level image.
 *      Filter size is ( half_filter_width * 2 ) + 1 by ( half_filter_height * 2 ) + 1
 *
 *                white - brighter than local average intensity.
 *                black - darker than local average intensity.
 *                gray  - within standard_deviation_threshold
 *                        * standard deviation of local average intensity
 */

Pic * VPS_create_regions(Pic * p, VPS_RegionTable * rt);
//------------------------------------------------------
/*
 *      Create 16-bit region image from 3 gray-level image. Each distinct region is given
 *      a unique 16-bit ID in range 1, 2, ..., n where n is number of distinction regions.
 *      n is the only value returned in the RegionTable structure - required by
 *      VPS_explore.
 */
```

```
void VPS_explore(Pic * p, Pic * region_pic, int pixel_limit,  VPS_RegionTable * rt);
//---------------------------------------------------------------------------
/*
 *      Further details are collected on each region i.e. bounding box, number of pixels,
 *      original 3-gray level colour (why 3 gray-level parameter is supplied!) Regions with less
 *      than "pixel_limit" pixels are discarded - parameters rt and region_pic will be
 *      updated accordingly. Region IDs are now 16-bit numbers: 1,2, ...m where m <= n.
 *      Must call before  VPS_slim_graph() which uses the RegionTable data.
 */

/*
 * #################
 * # FIND TARGETS #
 * #################
 */

VPS_Graph * VPS_create_scene_RAG(Pic * region_pic, VPS_RegionTable * rt);
//-------------------------------------------------------------
/*
 *      A RAG is created from the region_pic image. The numbers of nodes is taken from the
 *      number of regions held in the RegionTable.
 */

void VPS_slim_graph(VPS_Graph * scene, VPS_RegionTable * rt);
//-----------------------------------------------------------
/*
 *      Nodes in scene are either white, black or gray. This routine modifies scene to
 *      remove all gray regions; then to link all black nodes together that were connected
 *      to each gray nodes and to link all white nodes together that were connected to each
 *      gray node. The RegionTable is used to determine the colour of each node.
 */

VPS_SceneTargetTable * VPS_graph_search
(
        Pic * region_pic, VPS_Graph *scene, VPS_Graph *key, VPS_RegionTable * rt
);
//-------------------------------------------------------------
/*
 *      Find the targets visible in the scene by graph searching and obtain their 2D
 *      positions. The region image is used to get accurate region centroids.
 */

/*
 * ##################
 * # OBTAIN MAPPING #
 * ##################
 */

VPS_RecognisedTargetTable * VPS_recognised_targets
(
        VPS_SceneTargetTable * tt, VPS_WorldTargetTable * wtt
);
//-------------------------------------------------------------
/*
 *      Compare the targets found in the scene with the targets known to be in the world.
 *      Collate 3D world position and 2D scene positions of corresponding targets.
 */

VPS_Mapping * VPS_obtain_mapping(VPS_RecognisedTargetTable * rtt, double rms_limit);
//-----------------------------------------------------------------------------------
/*
 *      Find the mapping (a 4 x 4 matrix) between the 3D and 2D target positions. The
 *      mapping also holds the number of targets used to establish the matrix, a kind of
 *      quality factor. The more targets found the more accurately the mapping can be
 *      determined. The rms_limit parameter is the maximum root mean square positional
 *      error in pixels - between measured and re-computed target positions within the
 *      scene. Basically, this is a check to determine whether the mapping has been
 *      correctly determined. If the error is greater than the limit, the presumption is
 *      that the system has failed and a null mapping is returned.
 */
```

```
int VPS_null_mapping(VPS_Mapping * map);
//-------------------------------------
/*
 *      Checks for a null mapping.
 */
int VPS_coplanar_mapping(VPS_Mapping * map);
//-----------------------------------------
/*
 *      Checks for a mapping derived from a coplanar target set
 */
int VPS_mapping_quality(VPS_Mapping * map);
//-----------------------------------------
/*
 *      Number of targets used to establish mapping
 */
VPS_Camera * VPS_get_position_and_orientation
(
        VPS_Mapping * map, VPS_CalibrationCoefficients * cc
);
//-------------------------------------------------------
/*
 *      Splits the mapping obtained into a camera calibration matrix and an exterior
 *      orientation matrix. The latter provides the transformation between the world's and
 *      camera's frame of reference. The former provides information purely about the
 *      camera. Note that if the mapping has been obtained from a coplanar set of targets,
 *      then this split into two matrices is impossible without using a previously computed
 *      camera calibration matrix, supplied in cc. cc should always be supplied - although
 *      a non-coplanar set of targets may be being used - any particular view may only see
 *      a coplanar subset.
 */

/*
 * ##################
 * # VERIFY MAPPING #
 * ##################
 */

void VPS_verify_position_and_orientation
(
        Pic * s, VPS_Camera * mat, VPS_RecognisedTargetTable * rtt, int colour
);
//-----------------------------------------------------------------------------
/*
 *      Draws representation of the targets found within an image, using the 3D -> 2D
 *      mapping established. If the original image is supplied, then the target drawings
 *      should overlay the detected targets, and provide visual confirmation that the
 *      system is working.
 */

void VPS_verify_scene_targets(Pic * pic, VPS_SceneTargetTable * TT, int colour);
void VPS_verify_mapping
(
        Pic * s, VPS_Mapping * map, VPS_RecognisedTargetTable * rtt, int colour
);
void VPS_find_unidentified_targets
(
        VPS_Mapping * map, VPS_WorldTargetTable * wtt,
        VPS_RecognisedTargetTable * rtt, double distance
);
//-----------------------------------------------------
/*
 *      Uses mapping to "guess" at unidentified targets. rtt is updated - and can then be
 *      used to obtain a more accurate position fix using more targets
 */

#include "allocate_vps.h"
```

Appendix H

Utility Software: the X-Simple Library

H.1 Motivation

Throughout the development of the image processing code there was a requirement to visualise images both as simply and efficiently as possible and without any system dependencies which could lead to failure or non-portability. The natural choice might have been OpenGL, but as explained previously OpenGL is not a complete specification and system-specific commands are always necessary to create a context (such as a window) into which OpenGL can render. In addition, OpenGL is neither optimised nor well-designed for image display, as the model for pixel processing is unnecessarily heavyweight. The author decided upon a X-Windows implementation as this is almost universally supported. Programming directly in X is very messy: for any output to be visible, interrupts have to be caught and the programmer must always be aware of the pixel-depth of the screen to ensure correct image display. In the latter case display code must be replicated n-times where n represents the expected number of display architectures on which the code may be executed. A colleague at Essex University, Mike Lincoln, had wrapped frame-grabbing up in a small and tidy interface called HVCI (Human-understandable Video Capture Interface), and the author realised that perhaps it was time to do the same for image display after many years of wishing to do so. The image display code under X was tidied up and an API was designed. The resulting library is called X-Simple because it presents a radically simplified interface to X. However, the abstraction is such that the only concept that it inherits directly from X is the concept of networked operation. It was felt valuable to retain this.

H.2 X-Simple Philosophy

The whole of X has been reduced to six core operations. The key operation is that of displaying an image in a window as quickly and as simply as possible, and it is here that much interface design and performance optimisation effort has been expended. There is a single simple display primitive:

```
x_display_pic(int window_ID, Pic * p)
```

which displays image data (given by `p`) into a window (given by `window_ID`). To make the code platform independent, `x_display_pic` will always do something sensible even if the bits-per-pixel of the image differs from that of the display.

The image structure and image data used are completely visible and accessible to the application programmer, so that there are no restrictions on how image data may be manipulated: this is **not** what X-Simple is designed to hide. What X-Simple does is to display the image reliably, but hides all the nastiness that takes place after the display image call has been made. The image formats used are taken directly from the HVCI. One extra format called `VIDEO_8_RGB`, which is an 8-bit colour format, has been added so colour images can be displayed even if 8-bit display hardware is in use. The single existing 8-bit format had been grayscale. A few routines have also been provided that allow drawing within an image before it is displayed. These routines permit simple and efficient annotations and manipulations independent of the image format. So for example, drawing a red line on a colour image which is displayed to the screen will appear red, no matter the format of the image or the graphics mode of the screen. This level of abstraction is precisely what native X-Windows fails to achieve, and an attempt has been made to hide these shortcomings from the X-Simple API. When an X display connection is opened, there is the opportunity to switch on several optimisations for that connection. Currently these are:

1. Use of the Direct Graphics Access (DGA) mechanism to access the screen;
2. Use of MIT's Shared Memory Extension (SM) to X;
3. Use of HERMES assembly routines for pixel format conversion.

Options 1 and 2 are mutually exclusive. X-Simple can be built with or without dependencies on the DGA, SM and HERMES libraries. The simplicity of the API is demonstrated by the paucity of essential functions documented in Table H.1; the corresponding 'C' API is shown in Section H.4. A

Essential	Convenience
5 window system functions	3 utility image handling functions
• open/close display • open/close window • give window a title	• create • make (when memory already allocated) • delete image
1 display primitive	6 simple utility functions to draw in image
• display image in window	• set/get pixel • filled rectangle • draw string • draw line • RasterOp

Table H.1: Summary of X-Simple features

C++ binding for X-Simple is available. The "X-Simple" library will compile under both a 'C' and C++ compiler but clearly, the C++ interface only gets built if a C++ compiler is used. The C++ API is only a simple wrapper around 'C' code to maintain portability of the source — though from the outside, of course, the library might just as well have been written entirely in C++.

H.3 Performance Issues

The single X-Simple display primitive is written in terms of the X function `XPutImage`, which is the only means to display bit-map data residing in the data-space of an application program. Optimising X-Simple corresponds to optimising the use of the `XPutImage` function and any requisite pre-processing. To enhance performance, it is worthwhile addressing the following questions:

(Q1) How quickly does `XPutImage` operate in comparison to writing directly to the screen memory oneself?

(Q2) How much of an overhead is there in writing to a window instead of the screen?

(Q3) The X-Simple display routine implicitly uses image format conversion software, if necessary. Is this format conversion software, written in C, fast enough or would hand-coded, (MMX-specific) assembler routines be better?

H.3.1 Performance Enhancing Mechanisms

Consider an application which grabs an image using the HVCI interface and displays it using X-Simple. The kernel-resident video grabbing software used by HVCI digitises the video signal into a kernel-resident memory buffer. The X-Simple display code places this image in the off-screen back-up memory for a window. Only when the window is updated will the image actually appear on screen. Two "unnecessary" memory copies are involved in the process. Can these be circumvented to enhance performance? The reason the image is placed up in window back-up memory is so that the windowing system can repaint the window correctly without involvement of the application *e.g.* if the window is moved. In fact, the biggest overhead is sending this image through the X-protocol layer as it travels from one program (the X-client) to another program (the X-server).

However, there is a mechanism in UNIX which allows two process running on the same machine to share memory, and there is a shared memory extension to X (SM) produced by MIT which uses this mechanism for displaying images more efficiently. Should every image that is to be displayed be memory mapped between client and server, or should the window-backup memory alone be memory-mapped? The trade-off between the two approaches will depend on the relative performance of setting-up and free-ing multiple areas of shared memory compared to a simple memory copy to a single area of shared memory. The latter approach was chosen for simplicity.

Alternatively, instead of going through X, it is possible to write to the screen directly using the Direct Graphics Access (DGA) library, which permits the mapping of the hardware screen into user space. The location of the window can be obtained from X, so the image can be placed "manually" in the appropriate position on the screen via DGA. Irrespective of DGA, X-Windows allows images to be displayed directly on the display (called the root window) so there may be performance benefits drawing to the root rather than a standard window.

Another approach would be to grab the image directly on-screen, but currently frame grabbing is limited to the two kernel image buffers provided: one buffer is read by the user application while the kernel is grabbing to the other. Naturally memory sharing and direct writing to the screen will only work locally. However, the HERMES library, which has been written specially to perform fast image format conversion, can be used to enhance performance when required for both local and remote execution.

-ROOT (+ROOT)		image depth												DGA
		8				16				24				
display depth	8	33.5	(36.2)	34.5	(35.7)	9.3	(11.9)	14.0	(21.1)	7.0	(8.8)	7.2	(9.2)	-
		36.1	(35.8)	74.0	(73.5)	16.7	(15.7)	34.7	(31.7)	5.6	(5.2)	19.8	(20.0)	+
	16	13.9	(14.3)	14.9	(15.2)	16.9	(29.0)	16.6	(21.5)	6.5	(7.0)	7.0	(7.3)	-
		16.4	(16.3)	17.0	(17.0)	16.9	(16.7)	36.6	(34.3)	5.1	(5.0)	18.5	(17.5)	+
	24	16.2	(16.8)	12.5	(12.7)	8.5	(9.1)	10.7	(12.2)	6.0	(6.4)	6.7	(7.2)	-
		17.0	(16.9)	16.0	(15.7)	12.1	(12.1)	26.2	(25.5)	5.4	(4.8)	18.6	(17.1)	+
HER		-		+		-		+		-		+		

Table H.2: Results 1: X-Simple image display rates (in frames per second)

H.3.2 X-Simple Performance Evaluation

Three full PAL images (576×768) were grabbed in 8-bit grayscale, 16-bit colour and 24-bit colour respectively and then were displayed repeatedly in three display modes 8-bit, 16-bit and 24-bit respectively. This gives 9 different trials. For the first experiment, the DGA library, the HERMES library and drawing to the root window were used/not-used in all possible combinations. This gives 8 results for each trial. The second experiment used the same 9 trials but examined the effect of using the MIT shared memory extension to X (SM). This is compared with writing directly to the screen (DGA) and X-Windows on its own (X). The SM implementation was the last modification made to X-Simple and so was tested later. By this stage, it had been established that whether the root window was used or not had little impact on performance so this aspect was factored out. Performance figures are quoted in frames-per-second. The measurements were taken using on a 266 MHz Pentium II running X-Windows under Linux, using a RAGE 3D graphics board. Note that the graphics board used was rather old technology and somewhat idiosyncratic. For example, the 24-bit X-display mode is actually a 32-bit hardware mode — so 24-bit image display still needs format conversion. The performance profile has been noted to be **very** different and generally faster on other machines.

The performance results are presented in Tables H.2 and H.3. In the first table each cell is indexed by 5 parameters: image depth, display depth, and whether DGA, the root window or HERMES is being used. In the second table each cell is indexed by 5 parameters: image depth, display depth, whether HERMES is being used, and which of standard X, DGA or SM is being used. Each of the nine trials must be considered independently as they each represent a different computational task. For each image type the best performance occurs unsurprisingly when no format conversion is necessary. The highest frame rate achieved is 74 fps for 8-bit image $\rightarrow$ 8-bit display. However, greater than

			image depth					
			8		16		24	
display depth	8	X	43.5	42.6	14.3	15.2	17.4	12.6
		DGA	37.4	76.5	16.1	17.1	15.6	14.6
		SM	54.6	55.3	22.8	18.2	21.8	14.4
	16	X	10.6	15.6	19.9	21.2	9.4	11.7
		DGA	17.0	35.0	17.6	36.2	11.4	26.3
		SM	14.1	24.4	26.4	27.0	11.8	16.0
	24	X	8.0	8.8	7.4	8.4	7.4	7.9
		DGA	5.7	20.7	5.5	19.2	5.6	18.8
		SM	12.3	14.1	11.6	13.3	10.9	12.4
HER			-	+	-	+	-	+

Table H.3: Results 2: X-Simple image display rates (in frames per second)

real-time performance (25 fps) is still achieved in some cases of format conversion.

The first set of findings shows that switching on DGA and HERMES individually can actually reduce performance in some instances, but the combination of the two in the same instance will actually increase performance greatly! This appears nonsensical, yet is quite reproducible. The conclusion is that the ways in which the 'C' memory copy routine `memcpy` (for the DGA case), `XPutImage` (for non-DGA case), and assembler (for the HERMES case) access memory are not on a level playing field. The best performance is generally always achieved by using -ROOT, +DGA and +HERMES. Note the DGA timings are independent of whether the root window is used because the display bypasses the X-Window system. The possible speed-up in some instances *i.e.* 16-bit image $\rightarrow$ 8-bit display by changing how things are done can be around a factor of 4. The conclusion is that the largest overhead in the display module by far is the `XPutImage` of X-Windows. It is only when this is circumvented (using DGA) that the true benefits of using HERMES can really be seen. In short, `XPutImage` is very inefficient, but X cannot be driven any faster. It is to be hoped that the implementation efficiency of `XPutImage` improves: it was very tempting indeed to incorporate HERMES within X. A HERMES copy runs faster than a `memcpy`! It is natural to assume that `memcpy` uses the fastest option available to it, but clearly it does not.

The second set of performance findings show that SM gives most of the performance benefits of DGA without the disadvantage of locking up the X-Windows display (too much!). Here performance generally is at least at 25 frames per second for full PAL, even if pixel format conversion has been necessary. With HERMES, DGA is faster than SM. One would expect DGA always to give the

fastest results. Without HERMES, SM is faster than DGA. This is surprising, and suggests the shared memory version of `XPutImage` (`XShmPutImage`) can get to the screen memory faster than a `memcpy`. This is even more surprising given that the shared memory implementation may do an "unnecessary" memory copy of the user's image into the shared memory buffer before attempting to display it. This is required as the shared memory implementation has only a single shared image buffer per window (rather than a more generous one per image or a more frugal one per display). The previous trials did reveal, however, that `memcpy`'s performance is decidedly questionable. To try to speed up performance one could place all images to be displayed in shared memory, but the overhead of this is uncertain and ultimately was not evaluated.

The results are a salutary lesson not to look into performance issues! The intention was to ensure that the display module was reasonably efficient. Instead, the findings reveal that the fundamental underlying operations `memcpy` and `XPutImage`, upon which there is a necessary dependence, are far from efficient themselves! No more performance optimisation was attempted. X-Simple's performance level is practical, supporting real-time video. Given the law of diminishing returns, the easiest next step would be to write a mini-windowing system completely from scratch using the exceptionally well-optimised HERMES RasterOp assembly code. The outcome would easily and by far outstrip X's performance.

H.3.3 Recommendations

There is little advantage in using the root window (though one case does give a 50% speed-up). It is the combination of DGA or SM and HERMES that can **really** provide the speed-up, which is unfortunate as the use of DGA locks-up the X-Windows system for the duration of the application run. HERMES does worse than the 'C' code where the image and display depths are the same, and for some reason in the 8-bit image $\rightarrow$ 24-bit display case where the HERMES code must not be particularly efficient. Actually, for the non-DGA case when the image depth is equal to the display depth, HERMES does an unnecessary image copy because HERMES has been left to do its own thing. When HERMES finds no format conversion is required a straightforward copy is executed. This is fine for DGA because HERMES can copy direct to the screen. But with no DGA, there is no need to do an image copy before using `XPutImage`. This case was trapped and the benchmarks re-run. However, there was no measurable difference in the results. Limitations discovered in HERMES were communicated back to the maintainer of the software. Where the HERMES bitmap conversion software operated slower

than the equivalent X-Simple code, the X-Simple source was also submitted. Though useful speed-ups of X-Simple can be obtained through the use of external libraries, these libraries are not installed by default so some extra overhead is involved in using them. There is an argument for using a non-accelerated version of the X-Simple library, and waiting for many of the performance tweaks to be built into X, which is where they should reside in the first place. Accordingly, various build options are presented for the X-Simple library.

H.4 X-Simple API

```
extern int x_initialise_display(char * display_name, int * arg_depth, int flags);

        // initialise X system
        // retrieves colour depth of display
        // flags contains bits to switch use of HERMES and DGA on
        // returns display_ID

extern int x_create_window(int display_ID, int width, int height, int use_root_window);

        // create a window
        // returns window_ID

extern void x_display_pic(int window_ID, Pic * result);

        // display image "sensibly" in window
        // may need to map between image depth and hardware depth

extern void x_close_window(int window_ID);

        // closes down a window

extern void x_close_display(int display_ID);

        // closes down X

extern void x_name_window(int window_ID, char * label);

        // label window
```

Appendix I

Utility Software: the `watch` Program

I.1 Introduction

A video display utility called `watch` was written which displays "live" the sequence of images coming from a video camera. `watch` was used in a "virtual presence" application that slaves the motion of a steerable video camera off the motion of the VR headset which itself displays the camera image. The other component of this application is a program called `steer` documented in Appendix J which couples the headset and camera.

I.2 X-Simple Verification

`watch` was also used as an initial verify of the operation of X-Simple and to benchmark it in an application context. It is appropriate here to discuss the image format capabilities of X-Simple as they are based on the requirement to display (grabbed) images in X. The video grabbing software captures video in 8-bit monochrome, 16-bit colour and 24-bit colour. X-Simple takes this and displays it on an 8, 16, 24 or 32-bit deep frame buffer. The various combinations require different translation functions to be written. In Table I.1 only the testable translations are represented, although the software was written to perform every possible combination. For example, a frame buffer depth of 24 is not shown, as the PC being used did not work in a genuine 24-bit mode.

Image	Frame Buffer	Process	Constraint
8	8	straight copy	colour map is grayscale
16	8	5:6:5 RGB → 3:2:3 RGB	colour map set up specially
24	8	8:8:8 RGB → 3:2:3 RGB	colour map set up specially
8	16	8-bit grey level → 5:6:5 RGB	R=G=B
16	16	straight copy	
24	16	8:8:8 RGB → 5:6:5 RGB	
8	32	8-bit grey level → 8:8:8:8 RGBA	R=G=B, A undefined
16	32	5:6:5 RGB → 8:8:8:8 RGBA	A undefined
24	32	8:8:8 RGB → 8:8:8:8 RGBA	A undefined

Table I.1: Verified format conversions

I.3 Performance

I.3.1 Results

Full sized PAL is displayed at 25 fps in all cases where the display depth is equal to the capture depth (default operation). If this is overridden pixel unpacking and packing slows the performance to around 10 fps on a 266MHz PC. In normal operation format changing should not be necessary, and it is only provided to ensure that all possible scenarios will produce meaningful output on the screen

Networked performance of `watch` was also studied. On a 10BaseT network, the window updating was slow and sluggish at around 2 frames per second. In contrast, running the application across a 100BaseT network almost supported 16-bit colour at 25 fps. In fact a scale (down) factor of 1.7 had to be used to give 25 fps. A scale factor of 1.5 gave 12.5 fps, (exactly half real-time: every other frame could be grabbed. At a scale factor of 1.6, it alternated between 12.5 fps and 25 fps: this was obviously the critical value. Table I.2 shows what is achievable for different image depths; the practical bandwidth calculation is given below:

$$
\begin{aligned}
\text{Full 16-bit PAL Data Rate} &= 576 \times 768 \text{ pixels} \times 2 \text{ bytes / pixel} \times 25 \text{ frames / s} \\
&= 22 \text{ MBytes / s} \\
\text{Achieved 16-bit Data Rate} &= \frac{\text{Full 16-bit PAL Data Rate}}{(1.6 \times 1.6)} \\
&= 8.6 \text{ MBytes / s}
\end{aligned}
$$

Colour Depth	PAL Data Rate	Achievable Real-time Size
24	33 MBytes / sec	0.5 PAL (half size full colour)
16	22 MBytes / sec	0.6 PAL
8	11 MBytes / sec	0.8 PAL (almost full size grayscale)

Table I.2: `watch` network performance

I.3.2 Conclusions

Using just the X-interface was surprisingly effective: real-time display performance was achieved locally, and even more surprisingly the networked performance was almost real-time too (given the appropriate connection). Using the DGA or SA interface to enhance non-networked efficiency does not seem to be necessary. However, real subsidiary development benefits were obtained by working on the DGA code first. This is because several bugs were found in X that probably could not have been understood or worked around, had the DGA code not been written. Talking directly to the hardware is much simpler than using images under X, vindicating the prior use of DGA.

I.4 Bugs Revealed in X

`XCreateImage` sets up the wrong value for the `bytes_per_line` element of the image structure for all display modes apart from 8-bit. This had to be "manually" supplied before the `XPutImage` code would work. This took a long time to find. On the main development machine, even though X says it is in 24-bit mode, the image data have to be in a 32-bit format to be drawn successfully to the screen. Insight into the problem was only provided when, in despair, the code was moved to a different machine to debug. The code worked perfectly! It was the use of DGA that revealed that the development machine does not have true 24-bit hardware. One might have expected the X-interface to hide this when working in 24-bit mode, but it does not and this is a bug. `XCreateImage` will not accept a depth parameter of 32 (as X thinks it is running 24-bit). Consequently, it was necessary to create a 24-bit image, and then change the structure members "manually" to turn it into a 32-bit one.

I.5 `watch` Documentation

`watch` captures the video stream from the steerable camera and displays it on the VR headset. It takes the following parameters:

`-h`	Help
`-r`	Use Root Window
`-d`	Debugging Info
`-f<scale>`	Scale Down Factor (2.0 is half PAL, *etc.*)
`-g<depth>`	Video Grabbing Depth (8,16,24 or 32)
`-dga<0/1>`	Use Direct Graphics Access (DGA) mode to access screen memory directly
`-sm<0/1>`	Use a shared memory window back-up buffer to speed up image display

The option of displaying to the root window, allows the video to be readily displayed on the i-glasses. `watch` works with X running in 8, 16, 24 or 32 bit mode. Note that 8-bit capture is in mono, 16, 24 and 32 bit captures are in colour. By default, `watch` will grab at the same depth as the current X display. This gives best performance as neither pixel unpacking nor packing need be performed. By default, shared memory and DGA operation will be switched off. The frame-rate; number of bits per pixel captured; number of bits per pixel displayed; and size of image captured (in relation to full PAL) are displayed in the title bar of the output window. If the root window is being used, then these performance statistics are printed to the standard output. `watch` inherits the restrictions of the system software it uses:

(1) For the root window in 8-bit mode, the colour map does not get set-up properly. The DGA implementation has no such restriction. Attempts to set up the root window colour map will cause all subsequently launched X programs to run in the root window. This is a bug in X!

(2) Grabbing in 32-bit mode does not currently work. This is a known bug in the underlying bttv drivers.

Appendix J

Utility Software: the `steer` Program

J.1 Introduction

A subsidiary utility called `steer` was written to use the orientation sensors on the i-glasses VR headset to control the position of a steerable video camera. By running a second utility in parallel called `watch` (which takes the video image from the camera and displays it on the headset see Appendix I) a full remote presence application is created. The combination of the hardware components was irresistible, but the application also provided a simple means of testing the headset driver and display codes. The headset driver code had just been finished and the camera control code has been written some time previously by Mike Lincoln at Essex University. The camera interface requires the passing of specific x, y and z numbers to control the pan, tilt and zoom settings of the camera respectively. The meaning of these numbers is not documented and Table J.1 redresses the situation by providing input parameter ranges and a corresponding estimate of output physical effect.

Parameter	Controls	Approximate Range
-860 $\leq$ int x $\leq$ 861	absolute yaw angle	-95 degrees $\rightarrow$ +95 degrees
-282 $\leq$ int y $\leq$ 283	absolute pitch angle	-20 degrees $\rightarrow$ +20 degrees
0 $\leq$ int z $\leq$ 1023	absolute zoom	up to 12$\times$

Table J.1: Parameters to control steerable video camera and approximate effect

J.2 Problem 1

The motor-controlled camera is not capable of the complete freedom of movement of the i-glasses. Pitch, yaw and zoom are limited by mechanical stops, and there is no roll. Driving the camera from the headset is a case of finding the best compromise. Let the extreme yaw positions of the camera be P_0 and P_1. There is a point where the headset is outside the range of movement of the camera (namely $\pi + \frac{P_0+P_1}{2}$) where the most accurate position for the camera will switch suddenly from P_0 to P_1 or *vice versa*, resulting in an unnatural and unwanted sweeping motion of the camera from side to side. This *gedanken* experiment indicates that accuracy may not be the best criterion to set up the mapping.

J.3 Problem 2

The rate of panning has to be reduced at high zoom, otherwise everything whips by in a blur! When the camera is at the upper end of the zoom range, several rotations of the headset around the vertical axis should equate to one rotation of the camera (ignoring for the moment that camera movement is limited). Since the yaw angle is only reported by the headset within the range -180°to 180°, how can it be established many complete rotations have been made by the user to set up this gearing relationship properly? This second *gedanken* experiment gives the very surprising result that more than a simple orientation device is required *i.e.* one that remembers rotations from the past.

J.4 Solution

The solution is to build a virtual yaw device on top of the headset, which "guesses" at the number of rotations of the headset, and returns an angular measure taking this into account. If two subsequent headset readings are more than 180°apart, then it is assumed a rotation has occurred and the virtual yaw angle is adjusted accordingly. For example, a virtual yaw angle of 780°, indicates a real yaw angle of 60°, and the fact that two rotations have occurred. This simple abstraction solves both the problems. This abstraction is appropriate for encoding the yaw angle in *all* AR applications. As the headset is not capable of measuring the full range of pitch and roll angles, it is impossible to encode these similarly with this particular input technology. The two traditional approaches to cope with limited movement are:

1. scaling the range of the sensors into the range of movement

The headset will always respond to user movement giving transparency of operation, but the predetermined gearing may not be ideal.

2. truncating the range of the sensors within the range of movement

   A suitable gearing factor may be chosen.

Both approaches have been tried. Tests suggests that 2 is more satisfactory than 1. In order to combine the benefits of 1 and 2 to our "virtual yaw device" an additional and optional enhancement has been provided. Say the headset goes off to the right but the camera cannot follow. In the headset display one might show a red arrow flashing pointing to the left to show the valid direction of movement. Subsequent motion of the headset to the left can then cause *pro rata* immediate camera movement: although this dynamically changes the registration of camera and headset. Tests indicated that this enhancement improved usability. Suitable values for the gearing were computed by first determining the actual magnification factor of the camera as a function of the zoom parameter, see Appendix N. How should the zoom of the camera be controlled? Well, there is one spare parameter from the headset sensors: the roll. The movement of the head from side to side could be used to control the zoom. Above a certain threshold angle, the tilt of the head could be used to control the rate of zoom: clockwise to zoom in and anti-clockwise to zoom out.

J.5 `steer` Documentation

A number of command line parameters may be provided to `steer` to allow experimentation with many different ways of mapping headset to camera movement:

Miscellaneous Parameters

`-h`

Display help information

Input Device Parameters

`-kx -ky -kz`

Use keyboard for controlling x (yaw), y (pitch) and z (zoom)

right/left	arrow keys	for controlling x
up/down	arrow keys	for controlling y
page up/down	keys	for controlling z

`-hx -hy -hz`

Use headset for controlling x, y and z. The zoom input from the headset consists of leaning to the right for zooming-in and leaning to the left for zooming-out. This is rather difficult to control, and is not recommended. Note it is possible to mix and match keyboard and headset control: a sensible combination is `-hx -hy -kz` . If both headset and keyboard are used for the same dimension, the result is not guaranteed but it will probably work.

Headset Gearing Parameters

Headset gearing can work in three modes: absolute, relative and velocity. Each dimension x, y and z can work in one and only one mode.

`-ax -ay -az`

Absolute mode for x, y, and z. Movement of the headset is measured from a calibration point (generally taken at start of day) and this absolute measure is used to control the camera position. By default, the headset works in relative mode *i.e.* the camera uses its current position and the change in position of the headset to work out its new position. Zooming-in in absolute mode, just on its own, will naturally cause the direction of view of the camera to change because it changes the gearing factors.

`-vx -vy -vz`

Velocity mode for x, y and z. Movement from calibration point controls velocity rather than absolute position.

`-sx -sy -sz`

Auto-calibration for these dimensions. Basically, as well as being clipped to the range of the camera the gearing of the headset is recalibrated so motion in the other direction (away from the "hard stop") will have immediate effect. This happens naturally anyhow in relative mode. `-sx` is recommended so rotation of the headset (in at least one direction) can always have some effect instead of being outside the clip range and therefore totally ignored. `-sy` is not recommended

because a datum for humans is looking at the horizon level. Using this parameter would upset the tie-up of camera and headset horizon levels.

Keyboard Gearing Parameters

`-lx -ly -lz`

Ignore zoom factor for keyboard controlling x, y, and z. Here a single keystroke will increment x, y, or z by one. This corresponds to the smallest change that the hardware can make and is a rather slow (but nonetheless accurate) way of steering the camera. Otherwise by default, each key stoke will move right/left/up/down or zoom in/out by approximately 10% of the screen size intelligently adapting to the current zoom setting. This is more generally useful.

J.6 `steer` Findings

The most generally useful operation is `steer -hx -hy -kx -ky -kz -sx -ay`. Here, the keyboard can be used to override the headset where required, and alone supplies the zoom factor control in a reasonably practical manner. x works in a relative manner, whereas y works in an absolute manner. At high zoom factors, headset control feels less intuitive and makes large movements of the camera difficult or impossible due to the high gearing ratio, so keyboard control is possibly more effective in this case. Of course, unlimited movement of the camera at high zoom factors can be achieved in velocity mode using `steer -vx -vy -hx -hy -kx -ky -kz` but this is not any more successful than using the keyboard.

Appendix K

Slaving a Camera off the VR Headset using `watch` and `steer`

K.1 Remote Execution

Both `steer` (described in Appendix J) and `watch` (described in Appendix I) have been written to support remote execution, allowing a virtual presence in a remote location. If the images are displayed in an X window, `watch` will automatically run on the remote machine and produce a local display. With `steer`, the part which talks to the headset should be run locally and the part which talks to the camera should be run remotely. This can be achieved by using the `dev2soc` utility (written for the GPS system and described in Appendix D) which connects an RS232 device to a network socket.

K.2 Latency Considerations

The headset either sends a continuous stream of yaw, pitch and roll angle data or the headset orientation can be sampled. Intuitively, the latter is the minimal latency approach. However, in practice latency performance may be the other way around either if the data stream can be easily absorbed or if there is a non-negligible overhead is sending the request message. The camera holds a two instruction buffer. Therefore, latency may be improved by calling the waitComplete() method of the camera class to ensure one move instruction has been completed before sending another. No formal measurements of lag in the system were taken, and none of the above possibilities for reducing latency were investigated but they are mentioned for completeness. Latency depends on the range of movement: smaller

movements have a lower latency. The camera takes perhaps a second to travel its complete range of movement, and typical moves occupy a small fraction of a second.

K.3 Conclusions

To produce a slaved headset/camera system that was to any degree useful, requires thorough experimental investigations into the behaviour of any algorithms linking the output of the headset to the input of the camera. It is particularly noteworthy, that most of the problems encountered by the user controlling the system were entirely unforeseen and emerged only during testing. A system of reasonably optimised utility has been produced. The two biggest drawbacks in the system are:

1. The mechanical inertia in steering the camera - this introduces a considerable lag in response.
2. The limited motion of the camera, much of the intellectual and software effort was spent reducing the impact of this short-coming.

Appendix L

Reconciling VPS and OpenGL

For an AR system to function, the position subsystem must pass 3D transformation matrices to the underlying visualisation subsystem. In practice, it is almost inevitable that the two subsystems will not work together initially due to mistakes and incompatible matrix formats. How can such a system be debugged, when an arbitrary matrix is not easily interpretable by a human? For a start, a "guesstimate" projection stage can be hacked in, which makes things appear roughly the right size. This isolates the problem of obtaining a correct OpenGL modelview matrix [68]. Debugging to obtain concordance of projection matrices between VPS and OpenGL is the second stage.

L.1 Debugging VPS and OpenGL Alignment

Here is a description of how the matrix formats of VPS and OpenGL were reconciled in practice.

Stage 1.

Turn the camera around its optical axis in steps of $90°$, each time aligned with each of the axes of the world coordinate system. Record the coordinate transform as reported by the VPS between world and camera frames of reference; work out the coordinate transforms by hand from first principles. These tallied. The VPS produced a mathematically correct result — why then was it not suitable for OpenGL?

Stage 2.

Suppose M is the coordinate transform from world to camera established by the VPS. Each of M, M', M^{-1}, and $(M^{-1})'$ was loaded in turn into OpenGL. None of these worked. The matrix incompatibility was more complicated than a simple inverse or transpose convention.

Stage 3.

The OpenGL function `gluLookAt` is an easy way of specifying the camera to world transformation by supplying:

1. eye position (eye)

   → where you are

2. position of any point in centre of field of view (centre)

   → where you are looking

3. position of camera's up vector (up)

   → which way is up (vector coming out of the top of your head)

`gluLookAt` is the key to resolving the situation because the interface is semantically intelligible, so it is the ideal place to observe arguments going from the real world (comprehensible and under control) to the OpenGL world (where things seem to be wrong but somehow must be self-consistently correct). The following should be true:

1. The origin in camera coords should be the eye position in world coords.

$$\boldsymbol{eye} = \boldsymbol{M}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

2. The z-axis in camera coords is the direction the camera is looking in world coords.

$$\boldsymbol{centre} = \boldsymbol{eye} + \boldsymbol{M}^{-1} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

3. y-axis in camera coords is up-axis in world coords.

$$\boldsymbol{up} = \boldsymbol{M}^{-1} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

However, suppling these values to `gluLookAt` does not give the correct solution. This is not surprising, given that the matrix $\boldsymbol{M}$ did not work earlier. However, with `gluLookAt` it is possible to change, and thereby check, the operation of the parameters individually.

Sub-stage (i)

Choose arbitrary eye position in middle of lab: $\boldsymbol{eye} = (2, 2, 2)$.

Look directly up at ceiling (*i.e.* $z = 0$) : $\boldsymbol{centre} = (2, 2, 0)$.

Lie with head facing away from door: $\boldsymbol{up} = (0, 1, 0)$.

$\rightarrow$ Verified there was a valid OpenGL view of the ceiling

Confirmed that hard-wired parameters to OpenGL give expected effect.

Sub-stage (ii)

Change eye to be directly below a particular target

$\rightarrow$ Verified expected target in centre of OpenGL view

Confirmed $\boldsymbol{eye}$ parameter of `gluLookAt` working correctly

Sub-stage (iii)

Angle camera slightly off vertical with target in centre of view

Change $\boldsymbol{eye}$ to VPS calculated position

Change $\boldsymbol{centre}$ to known centre of target

$\rightarrow$ Verified target in centre of OpenGL view

Confirmed $\boldsymbol{centre}$ parameter of `gluLookAt` working correctly

Sub-stage (iv)

Change $\boldsymbol{up}$ to $\boldsymbol{up} = \boldsymbol{M}^{-1} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ which should be correct.

→ Target in centre of view **but** oriented upside down!

The $\boldsymbol{up}$ parameter of `gluLookAt` is **not** working correctly.

Reassign $\boldsymbol{up}$:

$$\boldsymbol{up} = -\boldsymbol{M}^{-1} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Things now work correctly!

Stage 4

`gluLookAt` does not work as expected. Is this a bug in `gluLookAt`? A version of `gluLookAt` was written based on (not unreasonable) pseudocode from the 'Net. This did not produce visible output and indeed behaved differently from the OpenGL version.

Stage 5

The source code of `gluLookAt` was obtained to further investigate. The original test transformation matrix $\boldsymbol{M}$ established by the VPS had been:

$$\boldsymbol{M} = \begin{bmatrix} -0.219771 & 0.979393 & 0.077629 & -6.124824 \\ 0.960086 & 0.228124 & 0.114282 & -2.728932 \\ 0.093555 & 0.103086 & -0.990437 & 0.686653 \\ 0.000000 & 0.000000 & 0.000000 & 1.000000 \end{bmatrix}$$

The equivalent single matrix pushed onto the GL stack to give the same effect as

$$gluLookAt \left(\boldsymbol{M}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \boldsymbol{M}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} + \boldsymbol{M}^{-1} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, -\boldsymbol{M}^{-1} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \right)$$

was

$$\boldsymbol{M}' = \begin{bmatrix} -0.218299 & 0.972831 & 0.077110 & -6.083785 \\ -0.971322 & -0.208970 & -0.113424 & 2.618084 \\ -0.094229 & -0.099658 & 0.990550 & -0.707964 \\ 0.000000 & 0.000000 & 0.000000 & 1.000000 \end{bmatrix}$$

Comparing $\boldsymbol{M}$ (which one might expect to be correct) and $\boldsymbol{M}'$ (what OpenGL needs), one can see that the elements of rows two and three have to have their sign changed. Whether there is a bug in the implementation or interface of OpenGL's `gluLookAt` or whether the low level matrix representation within OpenGL is anomalous is less important, as a method has been discovered to communicate VPS matrices directly into OpenGL. Possibly an ancient bug may be propagating which is forcing these strange conventions. Perhaps it is merely just an odd convention, but given how much trouble it has caused a great many people, assuredly an unwise one. It has been reported that the `gluLookAt` function acts differently depending on what version of Mesa is being used. `gluLookAt` was a convenient mechanism for debugging, and can now be abandoned. To debug directly the loading of raw matrices directly into OpenGL would probably have been impossible.

L.2 Solution

The following code shows how to pass a VPS (and mathematically standard) transformation matrix to OpenGL. Note that the VPS matrices were first established to be correct in strict mathematical terms, before reconciliation with the OpenGL matrix format was established.

```
void install_openGL_matrix(float ** P)
{
	int r, c;
	int i;
	float t[16];
	for(i = 0; i < 16 ;++i) // OpenGL has a silly known transposing convention
	{
		c = i / 4;
		r = i % 4;
		t[i] = P[r+1][c+1];
	}
	glMultMatrixf(t);
}

main()
{
	int r, c;
	// matrix in C numerical recipe format
	float ** M;

	// my function to create matrix with rows as columns as layed out textually

	M = matrix16
	(
		-0.219771, 0.979393,  0.077629, -6.124824 ,
		 0.960086, 0.228124,  0.114282, -2.728932  ,  // row 2
		 0.093555, 0.103086, -0.990437,  0.686653  ,  // row 3
		 0.000000, 0.000000,  0.000000,  1.000000
	);

	// code to pass matrix into OpenGL
	for(r=2;r<=3;++r)
	{
		for(c=1;c<=4;++c)
		{
			M[r][c] = -M[r][c]; // invert rows 2 and 3
		}
	}
	install_openGL_matrix(M);
}
```

Appendix M

Calling Java from ‘C’

To integrate the VPS ‘C’ code into the shared virtual world demonstrator Java code, it was necessary to call Java from ‘C’. Surprisingly, no complete documentation or code examples could be found on the Internet to accomplish this. The sample code below addresses this deficiency. It consists of a makefile (“makefile”); the C++ calling code (“calling.cc”) and the Java called code (“called.java”); The makefile and the Java source are trivial. The hard work is knowing how to make the C++ code call the Java code. To this end, the C++ is densely documented. The steps are as follows:

1. Set some initialisation parameters for the Java Virtual Machine manually.
2. Obtain remaining initialisation parameters for Java Virtual Machine by system call.
3. Override supplied CLASSPATH initialisation parameter manually.
4. Start Java Virtual Machine.
5. Look up class names to give class handles.
6. Look up method names to give method handles.
7. Instantiate objects (with handle of appropriate constructor method) to give an object handle.
8. Invoke object methods.

   Object and method handle are first two parameters.

   Java parameters are supplied as extra arguments.
9. Stop Java Virtual Machine

```
::::::::::::::
makefile
::::::::::::::
JAVAINCS  = -I /usr/local/jdk117_v3/include                \\
            -I /usr/local/jdk117_v3/include/genunix
LIBDIR    = -L /usr/local/jdk117_v3/lib                    \\
            -L /usr/local/jdk117_v3/lib/i686/green_threads

all: called calling

called: called.java
        javac called.java

calling: calling.cc
        g++ -o calling  calling.cc $(JAVAINCS) $(LIBDIR) -ljava_g

::::::::::::::
calling.cc
::::::::::::::
#include <stdio.h>
#include <stdlib.h>
#include <jni.h>
#include <string.h>

main()
{
     char buf[1000];
     JavaVM * p_vm;
     JNIEnv * p_env;
     void * vm_args;
     JDK1_1InitArgs init_args;
     jclass jc;
     jmethodID methodID_0, methodID_1, methodID_2;
     jobject obj;

     /*
      * We have to initialise the initialisation parameters
      * for starting the Java virtual machine. Silly!
      */

     init_args.version = 0x00010001;
                         // One note says we need a value in here.
                         // By doing this, things at least work.

     vm_args = (void *) &init_args;
     JNI_GetDefaultJavaVMInitArgs(vm_args);

     /*
      * The classpath supplied to the initialisation parameters is a
      * system one, and takes no account of your own classpath. Change
      * default classpath so things picked up from current directory.
      */
```

```
        strcpy(buf,".:");
        strcpy(buf + 2, init_args.classpath);
        init_args.classpath = buf;

        /*
         * Create Virtual Machine
         */

        JNICreateJavaVM(&p_vm,&p_env,vm_args);

         /*
         * Look-up your Class Name.
         * Note "java/lang/String" is a good test
         * as this will ALWAYS be found.
         */
        jc = p_env->FindClass("called");

        /*
         * Look-up your Method Names. You need to supply as last argument
         * the specific type signature of the method you want.
         * To get the constructor method ID use name "<init>".
         * To instance class properly, constructor ID is needed.
         */

        methodID_0 = p_env->GetMethodID(jc,"<init>","()V");
                   // constructor method
        methodID_1 = p_env->GetMethodID(jc,"bland","()V");
                   // method with no args
        methodID_2 = p_env->GetMethodID(jc,"receiving_7","(DDDDDDD)V");
                   // method with 7 args

        /*
         * Instance Object, invoking particular constructor method.
         */

        obj  =  p_env->NewObject(jc,methodID_0); // variable argument call

        /*
         *  Call Methods: one with no args, one with 7 args
         */
        p_env->CallObjectMethod(obj, methodID_1);
                                                  // variable argument call
        p_env->CallObjectMethod(obj, methodID_2,1.1,2.2,3.3,4.4,5.5,6.6,7.7);
                                                  // variable argument call

        /*
         * Close Virtual Machine
         */

        p_vm->DestroyJavaVM();

}
```

```
::::::::::::::
called.java
::::::::::::::
public class called
{
        static
        {
                System.out.println("STATIC JAVA Initialisation\n");
        }

        called()
        {
                System.out.println("JAVA Initialisation\n");
        }
        void receiving_7
        (
                double x, double y, double z,
                double r1, double r2, double r3, double r4
        )
        {
                System.out.println
                (
                        "JAVA Receiving 7 " +
                        x + " " + y + " " + z + " " +
                        r1 + " " + r2 + " " + r3  + " " + r4 + "\n"
                );
        }
        void bland()
        {
                System.out.println("bland\n");
        }
}
```

Appendix N

Calibrating Zoom of Sony Camera

N.1 Relative Calibration

The yaw and pitch angles, and the zoom factor of the Sony video camera can be automatically controlled via an RS232 connection. The parameters are called x, y and z respectively. It is fairly clear that x and y control linear stepper motors, so the relationship between x and yaw, y and pitch is linear. However, the relationship between z and the zoom factor is not clear and is not described in the documentation and so had to be derived experimentally. To avoid complex measurements of field of view, the edge of a row of ceilings tiles perpendicular to the angle of view of the camera was chosen. The focal length of the camera was adjusted until an integer number of tiles fitted horizontally across the frame at this edge. The results are shown in Table N.1 and plotted in Figure N.1. The graph of n against z is a nice regular curve but not a straight line. Another plot was made of $\frac{1}{n}$ (magnification factor is directly proportional to focal length) against z, but this was still not a straight line. Given that the relationship between z and zoom is not simple, it proved necessary to move to a second order

No of Ceiling Tiles (n)	z - value
6	18
5	125
4	258
3	405
2	587
1	826

Table N.1: Zoom parameter (z) to make n ceiling tiles fit across image

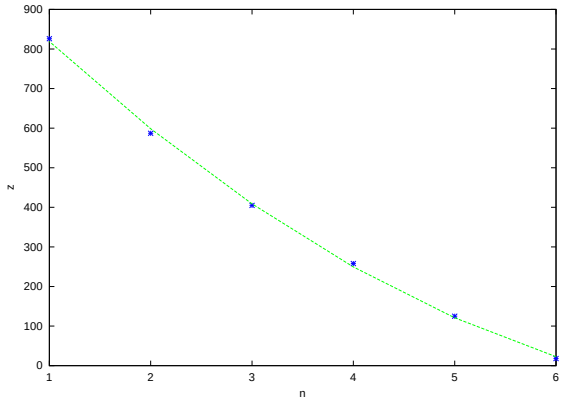

Figure N.1: Zoom parameter (z) to make n ceiling tiles fit across image

polynomial to obtain a close fit:

$$z = An^2 + Bn + C \quad \text{where} \quad A = 15.286, \quad B = -266.229 \quad \text{and} \quad C = 1069.800$$

N.2 Absolute Calibration

A formula has been found to link n to z. However, n (number of ceiling tiles) is a relative number and hardly a useful measure of the absolute zoom factor. To find a measure of the magnification factor, first z is altered until the size of the image on the VR headset is the same as reality. Here z is found to be around 362. The magnification factor is thus found to be:

$$M(z) = \frac{f^{-1}(362)}{f^{-1}(z)} \qquad \text{where} \qquad f(n) = An^2 + Bn + C$$

This equation translates z into a useful measure of camera zoom.

N.3 Conclusion

The measurements taken were hardly precise. However, the relationship established allows camera controlling software to take into account the zoom setting of the camera. The calibration was established for a virtual presence demonstration where usability rather than high accuracy is required. In the future, should a higher accuracy calibration be required a different technique will be employed such as the use of fiducials.

Bibliography

[1] M. Wheeler, *Roman Art and Architecture*. London: Thames and Hudson, second ed., 1973. ISBN 0 500 20021 1.

[2] M. V. Pollio, *De Architecura*. Como, first ed., 1521.

[3] R. T. Azuma, "A Survey of Augmented Reality," *Presence: Teleoperators and Virtual Environments*, vol. 6, pp. 355–385, August 1997.

[4] "The MIT Wearable Computing Web Page." `http://lcs.www.media.mit.edu/projects/wearables/`.

[5] W. Barfield and T. Caudell, *Fundamentals of Wearable Computers and Augmented Reality*. Lawrence Erlbaum Associates, first ed., 2001. ISBN 0-8058-2901-6.

[6] B. H. Thomas, W. Piekarski, and B. Gunther, "Using Augmented Reality to Visualise Architecture Designs in an Outdoor Environment," *International Journal of Design Computing*, vol. 2, 2000.

[7] T. Höllerer, S. Feiner, T. Terauchi, G. Rashid, and D. Hallaway, "Developing Indoor and Outdoor User Interfaces to a Mobile Augmented Reality System," *Computers & Graphics*, vol. 23, no. 6, pp. 779–785, 1999.

[8] R. Want and A. Hopper, "Active Badges and Personal Interactive Computing Objects," *IEEE Transactions of Consumer Electronics, February 1992*, February 1992.

[9] "Bluetooth: A Global Specification for Wireless Connectivity." `http://www.bluetooth.com/`.

[10] "Augmented Reality Review vs. Virtual Reality." `http://www.uk.infowin.org/ACTS/ANALYSYS/CONCERTATION/CHAINS/si/home/ch-sid/arvvr.html`.

[11] P. Milgram and F. Kishino, "A Taxonomy of Mixed Reality Virtual Displays," *IEICE Transactions on Information Systems*, vol. E77-D, December 1994.

[12] P. Jancène, F. Neyret, X. Provot, J. Tarel, J. Vézien, C. Meilhac, and A. Vérroust, "RES: Computing the Interactions between Real and Virtual Objects in Video Sequences," in *Second IEEE Workshop on Networked Realities*, (Boston, Massachusetts (USA)), pp. 27–40, Oct. 1995. `http://www-rocq.inria.fr/syntim/textes/nr95-eng.html`.

[13] H. Ishii and B. Ullmer, "Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms," in *CHI 97*, 1997.

[14] D. Drascic and P. Milgram, "Perceptual Issues in Augmented Reality," in *Proc SPIE, Stereoscopic Displays and Virtual Reality Systems III* (M. T. Bolas, S. S. Fisher, and J. O. Merritt, eds.), vol. 2653, pp. 123–134, January-February 1996.

[15] D. Drasic and J. J. Grodski, "Defence Teleoperation and Stereoscopic Video," in *Proc SPIE, Stereoscopic Displays and Applications IV*, vol. 1915, pp. 58–69, February 1993.

[16] D. Drasic, "Stereoscopic Video and Augmented Reality," *Scientific Computing and Automation*, vol. 9, pp. 31–34, June 1993.

[17] J. Hartman and J. Wernecke, *The VRML 2.0 Handbook, Building Moving Worlds on the Web*. Adison Wesley, first ed., 1996. ISBN 0-201-47944-3.

[18] D. Flanagan, *Java In A Nutshell, A Desktop Quick Reference*. 101 Morris Street, Sebastopol, CA 95472: O'Reilly & Associates, Inc., second ed., May 1997. ISBN 1-56592-262-X.

[19] A. Hopper, "Sensor-Driven Computing and Communications," in *IEE Seminar, Living Life to The Full with Personal Technologies*, 1998.

[20] J. Rekimoto, "The Magnifying Glass Approach to Augmented Reality Systems," in *International Conference on Artificial Reality and Tele-Existence '95*, 1995.

[21] C. Cruz-Neira, D. Sandin, and T. DeFanti, "Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE," in *ACM SIGGRAPH '93 Proceedings*, pp. 135–142, 1993.

[22] T. Starner, D. Kirsh, and S. Assefa, "The Locust Swarm: An Environmentally-Powered, Networkless Location and Messaging System," in *The First International Symposium on Wearable Computers*, pp. 169–170, October 1997.

[23] K. Finkenzeller and R. Waddington, *RFID Handbook: RadioFrequency Identification Fundamentals and Applications*. John Wiley & Sons, October 1999. ISBN 0470844027.

[24] A. Mulder, "Technical Report 94-1: Human Movement Tracking Technology," tech. rep., School of Kinesiology, Simon Fraser University, June 1994.

[25] "Position Tracking/Review of Tracking Techniques." `http://www.uk.infowin.org/ACTS/ANALYSYS/CONCERTATION/CHAINS/si/home/ch-sid/arvvr.html`.

[26] R. Hollands, *The Virtual Reality Homebrewer's Handbook*. Baffins Lane, Chichester, West Sussex, PO19 1UD, England: John Wiley & Sons, first ed., 1996. ISBN 0 471 95871 9.

[27] R. T. Azuma, *Predictive Tracking for Augmented Reality*. PhD thesis, University of North Carolina at Chapel Hill, February 1995.

[28] R. T. Azuma, "Tracking Requirements for Augmented Reality," *Communications of the ACM*, vol. 36, pp. 50–51, July 1993.

[29] S. Feiner, B. MacIntyre, T. Höllerer, and A. Webster, "A Touring Machine: Prototyping 3D Mobile Augmented Reality Systems for Exploring the Urban Environment," in *The First International Symposium on Wearable Computers*, pp. 74–81, October 1997.

[30] U. of North Carolina, "Wide-Area Tracking: Navigation Technology for Head-Mounted Displays," April 1998. `http://www.cs.unc.edu/~tracker/.`

[31] G. Welch, G. Bishop, L. Vicci, S. Brumback, and K. Keller, "High-Performance Wide-Area Optical Tracking The HiBall Tracking System," *Presence: Teleoperators and Virtual Environments*, vol. 10, pp. 1–21, February 2001.

[32] R. T. Azuma, "The Challenge of Making Augmented Reality Work Outdoors," in *Mixed Reality: Merging Real and Virtual Worlds*, pp. 379–390, 1999. ISBN: 3-540-65623-5.

[33] S. You, U. Neumann, and R. T. Azuma, "Hybrid Inertial and Vision Tracking for Augmented Reality Registration," in *Proceedings of IEEE VR '99*, (Houston, TX USA), pp. 260–267, March 1999.

[34] R. T. Azuma, B. R. Hoff, H. E. Neely, R. Sarfaty, M. J. Daily, G. Bishop, V. Chi, G. Welch, U. Neumann, S. You, R. Nichols, and . J. Cannon, "Making Augmented Reality Work Outdoors Requires Hybrid Tracking," in *First International Workshop on Augmented Reality*, (San Francisco, CA, USA), pp. 219–224, 1998.

[35] R. T. Azuma, B. R. Hoff, and H. E. I. Neely, "A Motion-Stabilized Outdoor Augmented Reality System," in *Proceedings of IEEE VR '99*, (Houston, TX USA), March 1999.

[36] S. You, U. Neumann, and R. T. Azuma, "Orientation Tracking for Outdoor Augmented Reality Registration," *IEEE Computer Graphics and Applications*, vol. 19, pp. 36–42, Nov/Dec 1999.

[37] S. R. Bible, M. Zyda, and D. Brutzman, "Using Spread Spectrum Ranging Techniques for Position Tracking in a Virtual Environment," in *Proceedings of the Network Realities '95*, 1995.

[38] K. Dowling, "Comp.robotics.FAQ," October 1996. `http://www.frc.ri.cmu.edu/robotics-faq/`.

[39] "Polhemus," 2000. `http://www.polhemus.com/home.htm`.

[40] "Ascension Technology Corporation," 2000. `http://www.ascension-tech.com/`.

[41] C. Verplaetse, "Inertial Proprioceptive Devices: Self-Motion-Sensing Toys and Tools," *IBM Systems Journal*, vol. 35, no. 3&4, pp. 639–650, 1996.

[42] A. Kelly, "Modern Inertial and Satellite Navigation," Tech. Rep. CMU-RI-TR-94-15, The Robotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, May 1994.

[43] S. Persa and P. Jonker, "Hybrid Tracking System for Outdoor Augmented Reality." Pattern Recognition Group, Technical University Delft.

[44] H. Weinberg, "Using the ADXL202 in Pedometer and Personal Navigation Applications," tech. rep., Analog Devices, ONE Technology Way, P.O. BOX 9106, Norwood, Massachusetts.

[45] G. Strang and K. Borre, *Linear Algebra, Geodesy, and GPS*. Wellesley Cambridge Press, first ed., 1997. ISBN: 0961408863.

[46] C. Rizos, “Principles and Practice of GPS Surveying,” 1999. http://www.gmat.unsw.edu.au/snap/gps/gps_survey/principles_gps.htm .

[47] *RTCM Recommended Standards for Differential Navstar GPS Service*. 655 Fifteenth Street, NW, Suite 300, Washington, D.C. 20005 U.S.A.: Radio Technical Commission For Maritime Services Special Committee No. 104, January 1994. Version 2.1.

[48] “Sam’s GPS Software Pages.” http://callisto.worldonline.nl/∼samsvl/software.htm.

[49] *NMEA 0183 Standards Document*. Seven Riggs Avenue, Severna Park, MD 21146, USA, version 3.0 ed., July 2000.

[50] P. Teunissen, P. de Jonge, and C. Tiberius, “Performance of the LAMBDA Method for Fast GPS Ambiguity Resolution,” *Navigation*, vol. 44, no. 3, pp. 373–383, 1998.

[51] “Ambiguity Resolution,” 1998. http://www.geo.tudelft.nl/mgp/lambda/index.html.

[52] *User’s Manual Allstar P/N 220-600944-00X*. 500 Dr. Frederik Philips Boulevard, Saint-Laurent, Quebec, Canada HAM 2S9: Canadian Marconi Company, June 1998. Publication No. 1200-GEN-0101.

[53] *Navstar GPS Space Segment/Navigation User Interfaces*. 2250 E. Imperial Highway, Suite 450, El Segundo, CA 90245-3509: Arnic Research Ccorporation, October 1993. ICD-GPS-200, Revision C, Initial Release.

[54] P. de Jonge, “Bibliography on GPS Ambiguity Resolution and Validation,” 1997. http://www.geo.tudelft.nl/mgp/people/paul/amb.html.

[55] “Proceedings of ION GPS 2000,” September 2000.

[56] G. Thomas, J. Jin, T. Niblett, and C. Urquhart, “A Versatile Camera Position Measurement System for Virtual Reality in TV Production,” in *Proceedings of IBC’97*, pp. 284–289, September 1997.

[57] R. Haralick, C. Lee, K. Ottenberg, and M. Nolle, “Review and Analysis of Solutions of the Three Point Perspective Pose Estimation,” *International Journal of Computer Vision*, vol. 13, pp. 331–356, December 1994.

[58] R. Jain, R. Kasturi, and B. Schunck, *Machine Vision*. Mc Graw-Hill, March 1995. ISBN:0-07-032018-7.

[59] J. Illingworth and J. Kittler, "A Survey of the Hough Transform," *Computer Vision, Graphics, and Image Processing*, vol. 44, no. 1, pp. 87–116, 1988.

[60] D. E. Knuth, *The Art of Computer Programming, Volume 1, Fundamental Algorithms*. Reading, MA, USA: Addison-Wesley, 1973. ISBN 0-201-03821-8.

[61] "2-Dimensional Bar Code." `http://www.adams1.com/pub/russadam/stack.html`.

[62] M. Dymetman and M. Copperman, "Intelligent Paper." Xerox Research Centre Europe, 6 Chemin de Maupertuis, 38340 Meylan, France, 1998.

[63] A. Glassner, "Penrose Tiling," *IEEE Computer Graphics and Applications*, vol. 18, pp. 78–86, July/August 1998.

[64] H. Kato, M. Billinghurst, and I. Poupyrev, "The MagicBook: Moving Seamlessly between Reality and Virtuality," *IEEE Computer Graphics*, vol. 21, no. 3, pp. 6–8, 2001.

[65] O. Faugeras, *Three-Dimensional Computer Vision, A Geometric Viewpoint*. The MIT Press, first ed., 1993. ISBN 0-262-06158-9.

[66] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C : The Art of Scientific Computing*. Cambridge University Press, 1988. ISBN: 0521431085.

[67] O. Faugeras, *Three-Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, Cambridge, Massachusetts, 1993.

[68] M. Woo, J. Neider, T. Davis, and D. Shreiner, *OpenGL Programming Guide*. Addison-Wesley, 3 ed., 1999.

[69] R. Young, *Visual Control in Natural and Artificial Systems*. PhD thesis, University of Surrey, January 2000.

[70] R. Tsai, "An Efficient and Accurate Camera Calibration Technique for 3D Machine Vision," in *Proceedings CVPR '86*, (Miami Beach, Florida), pp. 364–374, June 1986.

[71] I. Sinha, “The Evaluation of a Prototype Video Positioning System (VPS) and the Development of Tests to Characterise the Attributes of the System,” tech. rep., University of Essex, 2000. Final Year B.Eng Project.

[72] A. B. Houma, “The Evaluation of a Prototype Video Positioning System (VPS) and its Use Within an Augmented Reality (AR) Application,” tech. rep., University of Essex, 2000. Final Year B.Eng Project.

[73] R. Kalman, “A New Approach to Linear Filtering and Prediction Problems,” *Journal of Basic Engineering*, pp. 35–44, 1960.

[74] *RT3000 Specification*. Oxford Technical Solutions, 77 Heyford Park, Upper Heyford, Bicester, Oxfordshire, OX6 3HD, UK, 2001.

[75] P. Crummy, *City of Victory*. Colchester Archaeological Trust, first ed., February 1997. ISBN 1897719043.

[76] L. Piegl and W. Tiller, *The NURBS Book*. New York, NY: Springer-Verlag, 2 ed., 1996.

[77] J. A. Carson and A. F. Clark, “Multicast Shared Virtual Worlds Using VRML97,” in *Proceedings of the Fourth Symposium on Virtual Reality Modeling Language*, pp. 133–140, 1999. ISBN:1-58113-079-1.

[78] A. Schwerdtner and H. Heidrich, “Dresden 3D Display: A Flat Autostereoscopic Display,” in *Electronic Imaging / Photonics West 1998*, (San Jose, California), 1998.

[79] D. Johnston, “Generating Colour Autostereograms.” ESE Dept, Essex University, May 1998.

[80] A. Petersik, “The Stereoscope Applet,” 1999. `http://www.stereofoto.de/sapplet/sapplet_en.html`.

[81] A. S. Tanenbaum, *Modern Operating Systems*. New Jersey: Prentice Hall, 2 ed., February 2001. ISBN: 0130313580.

[82] N. J. Newman and A. F. Clark, “Sulawesi: A Wearable Application Integration Framework,” in *ISWC*, pp. 170–171, 1999.