User Tracking Methods for Augmented Reality Applications in Cultural Heritage

Gazi Erkan Bostancı

A thesis submitted for the degree of
Doctor of Philosophy

School of Computer Science and Electronic Engineering
University of Essex

November, 2013

Dedicated to
my loving wife Betül,
mum Cemile, dad Hüdaver and sister Burcu
as well as my grandfather Gazi and grandmother Güler who sadly
passed away before I could finish.

Abstract

Augmented Reality provides an entertaining means for displaying 3D reconstructions of ancient buildings *in situ* for cultural heritage. Finding the pose, position and orientation, of the user is crucial for such applications since this information will be used to define the viewpoint that will be used for rendering the models. Images acquired from a camera can be used as the background for such augmentations. To make the most out of this available information, these images can also be utilized to find a pose estimate.

This thesis presents contributions for vision-based methods for estimating the pose of the user in both indoor and outdoor environments. First an evaluation of different feature detectors is presented, making use of spatial statistics to analyse the distribution of the features across the image, a property that is shown to affect the accuracy of the homography calculated from these features.

An analysis of various filtering methods used for tracking was performed and an implementation of a SLAM system is presented. Due to several problems faced with this implementation, there is insufficient tracking accuracy due to linearity problems. An alternative, keyframe-based tracking algorithm is presented.

Continuing with vision-based approaches, Kinect sensor was also used to find the pose of a user for *in situ* augmentations making use of the natural features in the environment. Skeleton-tracking was also found to be beneficial for such applications.

The thesis then investigates combining the vision-based estimates with measurements from other sensors, GPS and IMU, in order to improve the tracking accuracy in outdoor environments. The idea of using multiple models was investigated using a novel fuzzy rule-based approach to decide on the model that results in improved accuracy and faster convergence for the fusion filter.

Finally, several AR applications are presented that make use of these methods. The first one is for *in situ* augmentation for displaying historical columns and augmenting users, the second is a virtual visit to an ancient building and the third is a game which can also be played inside the augmentation of the building in the second application.

Acknowledgements

I would like to extend my gratitude to my supervisor Dr. Adrian F. Clark for his insightful suggestions, constant help and support during my PhD. I would like to thank to Prof. Huosheng Hu, Prof. Dongbing Gu and Prof. Klaus McDonald-Maier for their invaluable suggestions in supervisory board meetings.

I would like to thank Ankara University for supporting my research and my colleagues in Computer Engineering Department for their support.

I would like to thank my colleagues Dr. Nadia Kanwal and Dr. Shoaib Ehsan for fruitful discussions; Robin Dowling and Kevan Wilding for technical support; Ayşenur Bilgin, Panitnat Yimyam, Felix Ngobigha, family members of Dr. Kanwal and Dr. Ehsan, Nicholaos Vastardis, Dr. Andreas Bontozoglou, Mahin Abbasipour, Behrooz Koohestani, Ahmad Alzahrani and Ahmed Mohamed for being great friends.

I am indebted to my family Cemile, Hüdaver and Burcu Bostancı for their constant support and love all these years. I am always grateful for having wonderful parents and a lovely sister. I also would like to thank to my inlaws Şükran and Şahabettin Eser and all my relatives for their support.

Finally, I also cannot express my gratitude to my other half, Betül, for her constant and never-ending love, unlimited patience and great support throughout years.

CONTENTS

LIST OF ABBREVIATIONS

2D 2-dimensional.

3D 3-dimensional.

ANOVA ANalysis Of VAriance.

AR Augmented Reality.

BRIEF Binary Robust Independent Elementary Features.

CAD Computer Aided Design.

CCD Charge Coupled Device.

CML Concurrent Localization and Mapping.

CMM Constant Motion Model.

CMOS Complementary Metal Oxide Semiconductor.

CPU Central Processing Unit.

CSG Constructive Solid Geometry.

DLT Direct Linear Transformation.

DoF Degree of Freedom.

DoG Difference of Gaussians.

DSP Digital Signal Processor.

E-PnP Efficient Perspective-n-Points.

EBR Edge-based Regions.

ECEF Earth-Centred Earth-Fixed.

EKF Extended Kalman Filter.

FAMM Fuzzy Adaptive Motion Model.

FAST Features from Accelerated Segment Test.

FAST-ER Features from Accelerated Segment Test-Enhanced Repeatibility.

FLANN Fast Library for Approximate Nearest Neighbours.

FLC Fuzzy Logic Controller.

FOV Field Of View.

FPS First Person Shooter.

fps frames per second.

GFT Good Features to Track.

GIS Geographical Information System.

GP Genetic Programming.

GPS Global Positioning System.

GPU Graphical Processing Unit.

HarAff Harris-Affine.

HarLap Harris-Laplace.

HCI Human Computer Interaction.

HesAff Hessian-Affine.

HesLap Hessian-Laplace.

HMD Head Mounted Display.

IBR Intensity-Extrema-based Regions.

IMU Inertial Measurement Unit.

IR Infra-Red.

JCBB Joint Compatibility Branch and Bound.

KF Kalman Filter.

KLT Kanade-Lucas-Tomasi.

LED Light Emitting Diode.

LoG Laplacian of Gaussian.

LSD Least Significant Difference.

MARG Magnetic, Angular Rate and Gravity sensor.

MEMS Micro Electro Mechanical System.

MutEx Mutual Exclusion object.

NCC Normalized Cross-Correlation.

NCFM Network Coupled Feature Maps.

NEF Nikon Electronic Format.

NN Nearest Neighbour.

OpenNI Open Natural Interaction.

ORB Oriented BRIEF.

PF Particle Filter.

PnP Perspective-n-Points.

PPM Portable Pixel Map.

RAG Region Adjacency Graph.

RANSAC RANdom SAmple Consensus.

RFID Radio Frequency IDentification.

RGB Red, Green and Blue.

SAD Sum of Absolute Differences.

SDL Simple DirectMedia Layer.

SFM Structure From Motion.

SFOP Scale invariant Feature OPerator.

SIFT Scale Invariant Feature Transform.

SLAM Simultaneous Localization and Mapping.

SOM Self-Organizing Map.

SSD Sum of Squared Differences.

SURF Speeded-Up Robust Features.

SUSAN Smallest Univalue Segment Assimilating Nucleus.

SVD Singular Value Decomposition.

UAV Unmanned Aerial Vehicle.

USAN Univalue Segment Assimilating Nucleus.

USB Universal Serial Bus.

VPS Video Positioning System.

VR Virtual Reality.

VRML Virtual Reality Modelling Language.

WAV WAVeform audio file.

XOR Exclusive OR.

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

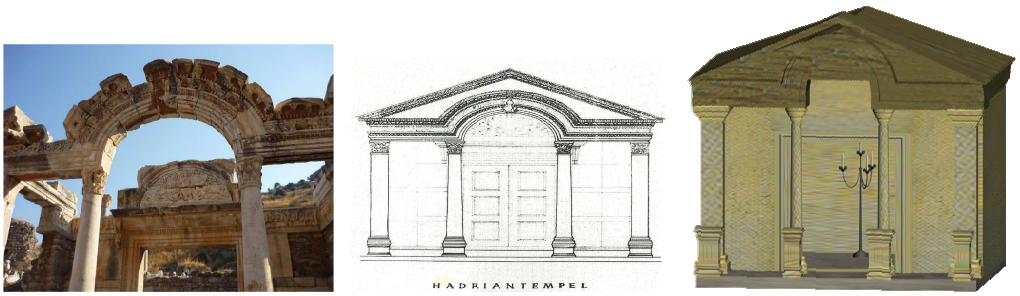
Technologies presented every day have brought us into a state of mind which is more demanding and more difficult to satisfy. We cannot be satisfied with the things we have, we do and, more recently, we see! Our eyes look for additional content in the same environment we see every day. This demand provides a powerful input to research as new challenges to be addressed and unknown areas to be investigated. As new solutions for existing problems are developed and as researchers share their future prospects with public, the demand is even more increased because the results will be combined with the future dreams of people, forming a cycle[1].

Augmented Reality (AR) is a technology which attempts to fulfil the demand for additional content in everyday scenes by inserting virtual (usually synthetic, though live videos can also be inserted) objects over the content we would normally

[1] http://www.creative-science.org/wp-content/uploads/2012/01/CSf-P_The-Creative-Science-Cycle1.pdf

see, in the form of overlays, typically using a Head Mounted Display (HMD). Having a wide spectrum of application areas, ranging from industrial maintenance and repair tasks to military training and entertainment, AR has already shown that it has much potential.

This research concentrates on AR for cultural heritage. The principal reason for performing AR reconstructions in cultural heritage is that the owners of sites are usually reticent to permit physical reconstructions *in situ* so that the archaeology remains undisturbed for future generations. Developments in multimedia technology facilitate the learning experience in cultural heritage with the aid of improved user interaction methods. Developed models (*e.g.* Figure 1.1) or virtual tours of reconstructions of archaeological sites provide entertaining means of learning.



(a) Temple in ruins (b) A reconstruction from [1, 2] (c) 3D model based on (a) and (b)

Figure 1.1: Views from the Hadrian Temple in Ephesus, Turkey

However, *ex situ* reconstructions such as models and movies are difficult to visualize in the context of the archaeological remains. AR reconstructions can be produced *in situ* with minimal physical disturbance, an attractive property, even though they may take a significant time to develop due to a number of challenges.

An important challenge is producing a realistic output image in which the

inserted objects are coherent with the real-world scene. Locating and tracking the user accurately in the environment [3] is key to this process since the user position and orientation determine the perspective information required for augmenting the scene. The process of tracking the user position is relatively easy if the user stands in a fixed, known position in a structured environment, looking at a fixed scene. Tracking the user becomes much more difficult if he or she is able to move freely in an outdoor environment where there is no limit for possible movements. This thesis will show that this is best achieved using a combination of position and orientation estimates from several sensors, using efficient algorithms to perform user tracking in real-time.

1.1 Motivation and Approach

The dream feeding and motivating the research presented in this thesis is a "Total Augmentation Paradigm" which can be illustrated as Figure 1.2 and conceptualized with the two creative science prototype stories given in [4].

The main focus of the thesis is on finding the position of the user since this is a crucial part for such a system. This will be handled using a tracking system that comprises a camera, a Global Positioning System (GPS) receiver and an Inertial Measurement Unit (IMU). Each of the sensors are low cost and public grade, in other words the accuracy of the sensors are not satisfactory for the application when they are used alone. For this reason, sensor fusion will be implemented to achieve more accurate position and orientation estimates.

The camera used in the system has a the task of acquiring images of the scene, which will be used for two purposes. Firstly, they will be used by the vision-based algorithm developed in order to provide estimates for the user position

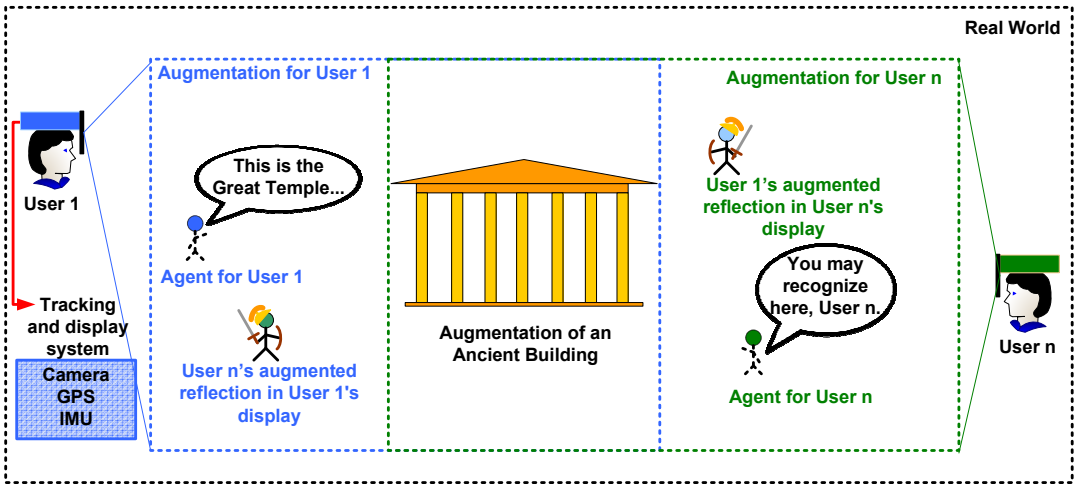


Figure 1.2: Prototype system for a "Total Augmentation Paradigm". Each user carries a tracking and display device and views the reconstruction of an ancient building. Users appear augmented in a clothing appropriate for the age when the building was available to other users of the system. Games inside such a system may make it even more interesting and entertaining. In addition, each user has an individual AR agent [4,5] that helps them by providing information about the history of the building.

using algorithms from the computer vision domain. The literature provides many different methods for vision-based pose estimation and the thesis makes use of some of these methods with a novel approach. Secondly, the images will be used to produce the final output where augmentations can be overlaid on the original scene.

The GPS receiver and IMU will aid the camera for positional estimates since the prototype system is intended to be used outdoors. Positional estimates from these sensors will be unified with the vision-based ones in order to produce more accurate results. A second task for the IMU is to provide orientation estimates using a recent robust method from the literature [6].

The integration of these several sensors was implemented following a multi-threaded approach to tackle the real-time challenge and using multiple fuzzy-adaptive motion models for improved accuracy.

Use of Microsoft's Kinect sensor was also investigated for producing *in situ* augmentations for historical columns, and skeleton-tracking features for augmenting participants of the system.

1.2 Contributions

The major contributions presented in this thesis are as follows:

1. A metric for evaluating the coverage of image feature points was developed and the effect of feature coverage on homography estimation was analysed with a focus on image stitching applications. (Chapter 3) [7, 8]

2. A vision-based tracking method was developed using two-view geometry with an automated keyframe extraction algorithm for video sequences. (Chap-

ter 4) [9]

3. A parallel algorithm was developed for finding planar features in a scene using the Kinect sensor. (Chapter 5) [10]

4. An *in situ* augmentation algorithm was developed using the depth values from the Kinect sensor as well as its skeleton tracking features. (Chapter 5) [11]

5. An algorithm for integrating position and orientation estimates from a camera, GPS receiver and an IMU was developed. (Chapter 6)

6. Use of fuzzy-adaptive motion models was investigated in order to reduce the filter error and handle uncertainty better within a Kalman filtering framework. (Chapter 6)

7. Three cultural heritage applications were developed. The first application augments scene objects with synthetic columns and users with synthetic clothing. The second is for a virtual visit to an ancient building and the third is a simple AR game. (Chapter 7) [9, 11]

1.3 Publications

Parts of the contributions presented in the thesis and other research outputs have been published or are under review:

1. **Bostanci, E.**, Kanwal, N. and Clark, A. F., "Spatial statistics of image features for performance comparison," Image Processing, IEEE Transactions on, vol. 23, no. 1, pp. 153–162, 2013.

2. **Bostanci, E.**, Kanwal, N. and Clark, A. F., "Augmented Reality Applications for Cultural Heritage Using Kinect", submitted to ACM Journal on Computing and Cultural Heritage.

3. Kanwal, N., **Bostanci, E.**, and Clark, A. F, "Matching Corners Using the Informative Arc," to appear in IET Computer Vision.

4. **Bostanci, E.**, Kanwal, N., Ehsan, S. and Clark, A. F. , "User Tracking Methods for Augmented Reality". International Journal of Computer Theory and Engineering, 5, 1, pp.93–98. ISSN 1793-8201, 2013.

5. **Bostanci, E.**, Kanwal, N. and Clark, A. F., "Kinect Derived Augmentation of the Real World for Cultural Heritage," Proceedings of UKSIM'13, Cambridge, UK, 2013.

6. Kanwal, N., **Bostanci, E.**, and Clark, A. F, "Kinect Aided Navigation System for Visually Impaired People," Proceedings of the Workshop on Recognition and Action for Scene Understanding (REACTS 2013), York, UK, 30-31 August 2013.

7. **Bostanci, E.**, Kanwal, N. and Clark, A. F., "Extracting Planar Features From Kinect Sensor," Proceedings of CEEC'12, Colchester, UK, 2012.

8. **Bostanci, E.**, Clark A. F., Kanwal, N., "Vision-based User Tracking for Outdoor Augmented Reality," Proceedings of the 17^{th} IEEE Symposium on Computers and Communication, Cappadocia, Turkey, 2012.

9. Kanwal, N., **Bostanci, E.**, and Clark, A. F., "Describing Corners using the Angle, Mean Intensity and Entropy of Informative Arcs," Electronics Letters, vol. 48, no. 4, pp.209–210, 2012.

10. **Bostanci, E.**, Kanwal, N. and Clark, A.F, "Feature Coverage for Better Homography Estimation: An Application to Image Stitching," Proceedings of IWSSIP'12, Vienna, 2012.

11. **Bostanci E.**, Clark A. F., "Living the Past in the Future", 2^{nd} International Workshop on Creative Science, pp.167–172, Nottingham, UK, 2011.[2]

12. Kanwal, N., Ehsan, S., **Bostanci, E.** and Clark, A. F., "Evaluating the Angular Sensitivity of Corner Detectors," Proceedings of the IEEE International Conference on Virtual Environments, Human-Computer Interfaces, and Measurement Systems (VECIMS), Ottawa, Canada, September 2011.

13. Kanwal, N., Ehsan, S., **Bostanci, E.** and Clark, A. F., "A Statistical Approach for Comparing the Performances of Corner Detectors," Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, Victoria, B.C., Canada, August 23-26, 2011.

14. **Bostanci, E.**, Kanwal, N., Ehsan, S. and Clark, A. F., "Tracking Methods for Augmented Reality," Proceedings of the 3^{rd} International Conference on Machine Vision (ICMV), Hong Kong, December 2010.

15. Ehsan, S., Kanwal, N., **Bostanci, E.**, Clark, A. F. and McDonald-Maier, K. D., "Analysis of Interest Point Distribution in SURF Octaves," Proceedings of the 3^{rd} International Conference on Machine Vision (ICMV), Hong Kong, December 2010.

[2]Received Intel's free registration prize.

1.4 Thesis Outline

The structure followed in the thesis is briefly summarized in Figure 1.3 where the logical sequence of the chapters can be seen.

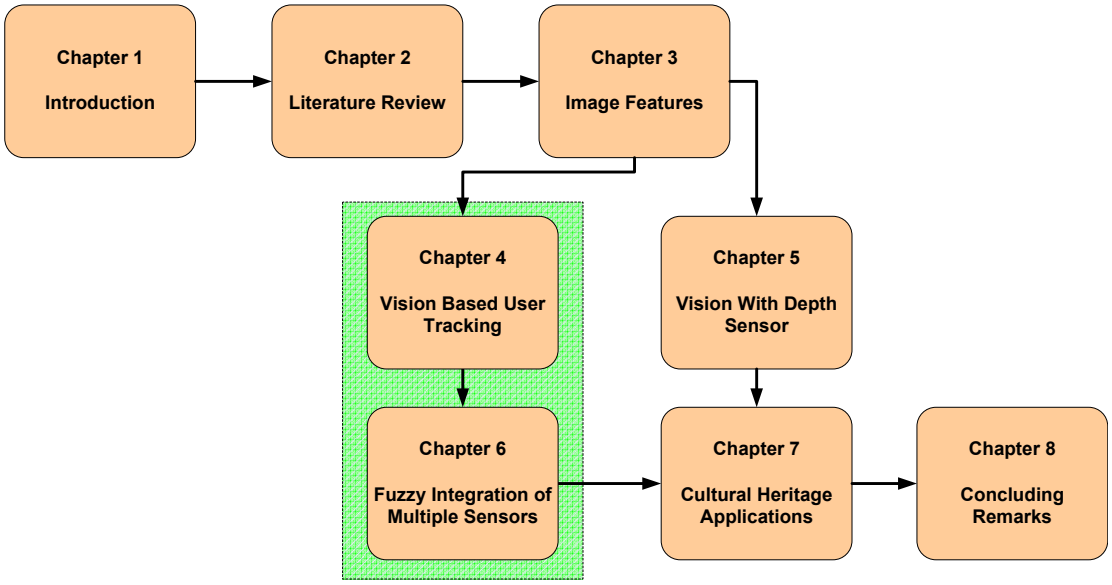


Figure 1.3: Relationships between chapters

Chapter 2 presents a review of the literature giving a broad analysis of the previous work on tracking methods for AR.

Chapter 3 presents the first contribution of the thesis as an evaluation of different feature detectors based on their coverage on the input image using a robust metric that has not been used in the vision domain previously. This important measure is shown to affect the homography matrix (calculated using these features) which is used in many different vision-based applications such as image stitching or pose estimation.

Chapter 4 presents the second contribution of the thesis, a vision-based user tracking method based on keyframes. The chapter starts with background information on camera calibration and filtering methods then proceeds with an ini-

tial attempt using a popular monocular Simultaneous Localization and Mapping (SLAM) method and finally explains the proposed approach.

Chapter 5 presents experimental work and contributions using the Kinect sensor. The first contribution is a parallel algorithm for finding planar features in a scene using the explicit definition of a plane. The second contribution is for detecting column-like objects in a scene so that they can be augmented with synthetic columns in a cultural heritage context. The chapter also discusses the skeleton tracking features of the sensor and its software which can be used to augment users according to the fashion trends of a particular age.

Chapter 6 combines motion estimates from a GPS and an IMU with the estimate obtained using the method described in chapter 4 as its first contribution. In addition to this, the use of multiple motion models were investigated using a fuzzy rule-base in order to capture the user's motion more accurately and reduce filter errors.

Chapter 7 presents three applications, one of them using the *in-situ* augmentation algorithm presented in chapter 5 and two using the tracking system developed in chapter 6 as the final contribution of the thesis.

Chapter 8 draws conclusions by giving a brief summary of contributions and suggests future research topics as further improvements to the algorithms and systems developed in this thesis.

CHAPTER 2

LITERATURE REVIEW AND BACKGROUND

This chapter presents a broad survey of the previous work on the topics that are covered in the thesis. The discussion starts with a review of the computer vision (section 2.1) methods which are used to acquire information about the scene being captured using a camera. Topics include cameras and their calibration, methods for extracting useful information from images and representing this information and finally approaches for finding camera pose using captured images.

Once an estimate of the camera pose is found, it can then be refined by employing different filtering methods (section 2.2) for estimation problems in which data are unreliable and noisy. The discussion continues with use of these filters in techniques used for robot localization in section 2.3.

A filtering method can make use of various models for updating itself that can

be selected after a decision making process. Fuzzy logic, which allows decision making to be performed with computers in a similar way to humans, is described in section 2.4, followed by a discussion of the statistical methods used in the thesis for an evaluation in section 2.5.

After describing these underlying principles, application examples of AR are presented, along with how it is used in cultural heritage applications in section 2.6. This section will be followed by a review of user tracking methods for AR in section 2.7, including different approaches for indoor and outdoor environments using different sensors. Then, the shortcomings of current tracking systems and approaches are presented in section 2.8.

Finally, the chapter is concluded with guidelines for developing a user tracking system, considering the principles laid in the previous sections, in section 2.9.

2.1 Computer Vision

Vision is an important sense for humans since it allows them to understand the structure of their environment. This process of inferring the spatial relationships (*i.e.* 2-dimensional (2D) positions and perspective order) between the objects in the surrounding can be described in two stages. First, the reflected light from the objects in the environment must be sensed through a sensor (the eyes), then it must be interpreted by a processing mechanism (the brain) to make sense of the surroundings.

The process becomes harder if the environment is not static *i.e.* constantly changing in terms of viewpoints (*e.g.* self-motion), dynamic content (*e.g.* moving objects) and lighting conditions (*e.g.* day/night, shadows, *etc.*). Fortunately, our brains dedicate half of the cerebral cortex, the outer layer of the brain, for this

processing [12] and can perform the necessary 'calculations' to understand these spatial relationships instinctively.

Trying to emulate the same functionality with computers instead of the human brain using cameras as sensors is harder, for a number of reasons:

1. Cameras act as a very limited sensor since they do not include any dedicated structures for sensing shapes and colours separately such as the rod (vision in low light) and cone (coloured vision) cells in eyes. The only input for a vision system from a camera is a 2D array of intensity or colour values.

2. The human brain is much more powerful than current computing resources. One second of neuronal network activity can be simulated using a supercomputer known as "The K" with $82,944$ processors in 40 minutes[1]. Costs for such systems are at the order of billions of dollars.

3. Apart from the two hardware challenges mentioned in the items above, the current literature does not present a complete system of algorithms to carry out the tasks which our brains can perform intuitively.

Having accepted the practical challenges and current technological limitations as harsh facts of life, the next step is to have a look at what can be done with currently attainable resources, such as a modest desktop or a laptop computer. In fact, a wide range of applications can be found, from navigation systems for autonomous robots [13–15] to Unmanned Aerial Vehicles (UAVs) [16–18] or cars using motion analysis in detection and recognition tasks, including face recognition [19, 20], item recognition for manufacturing [21, 22] and tracking [23, 24], 3-dimensional (3D) scene reconstruction [25–27], video stabilization [28, 29] and adding special effects for movies [30].

[1] http://www.riken.jp/en/pr/press/2013/20130802_1/

Most of the applications given above require an understanding of the scene and finding spatial parameters for the camera, which is an involved process. Common approaches start with a camera calibration step, which aims to identify the internal parameters of the camera, and continues by finding and extracting useful bits of information called features from the images; and then calculating a signature or 'descriptor' for these features that is assumed to identify them uniquely. These descriptors are then used to establish correspondences between images, after which methods for motion estimation can be used to find spatial parameters such as position and orientation. The estimate can then be refined using a filtering method (section 2.2) or a more expensive process mentioned in section 2.1.3.

The following subsections explain some of the sensors, algorithms and methods that make such applications possible.

2.1.1 Cameras and camera calibration

A digital camera can be viewed as two components, the lens and the imaging sensor. Reflected light from objects pass through the lens and is then projected onto the sensor, which can be manufactured as Charge Coupled Device (CCD) or Complementary Metal Oxide Semiconductor (CMOS) device, both comprising of an array of sensors sensitive to light. These sensors convert the light into electrical signals which can be read out digitally for storage or processing.

This relatively complex imaging process is normally represented using an ideal pinhole camera model [25, 31, 32]. In this simple model, shown in Figure 2.1, the camera is modelled using a 3D position for the optical centre and a 2D image plane. The focal length of the camera is the shortest distance between the optical centre and the image plane. The projection of a 3D point can be obtained by

drawing a line from the optical centre through the image plane to the 3D point. The projection is found as the 2D location on the image plane.

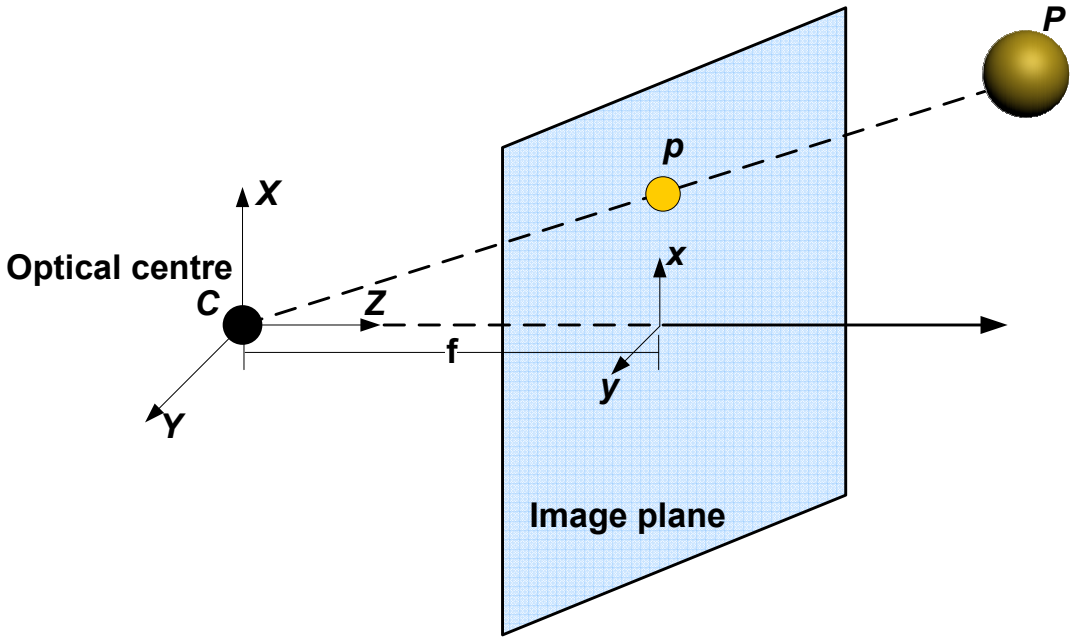


Figure 2.1: Pinhole camera model. C is the optical centre and f is the shortest distance from C to the image plane. P is a 3D point with its projection p on the image plane.

Unlike this theoretical representation, real-world cameras introduce distortion due to problems in the manufacturing process. For a more realistic representation, these distortion parameters should also be included in the projection model. The process for finding these parameters (as well as other internal parameters such as the focal length) is called camera calibration [33, 34]. There are dedicated toolboxes for this purpose (*e.g.* [35]) which can be used to find the distortion parameters as long as an image sequence acquired with that camera is provided.

A camera is used for vision-based user tracking algorithm and the camera calibration is performed as described in chapter 4.

Kinect as an RGB and depth sensor

Microsoft's Kinect sensor provides depth information as well as conventional Red, Green and Blue (RGB) colour space images. The device, shown in Figure 2.2, includes an Infra-Red (IR) projector and sensor in addition to a conventional camera.

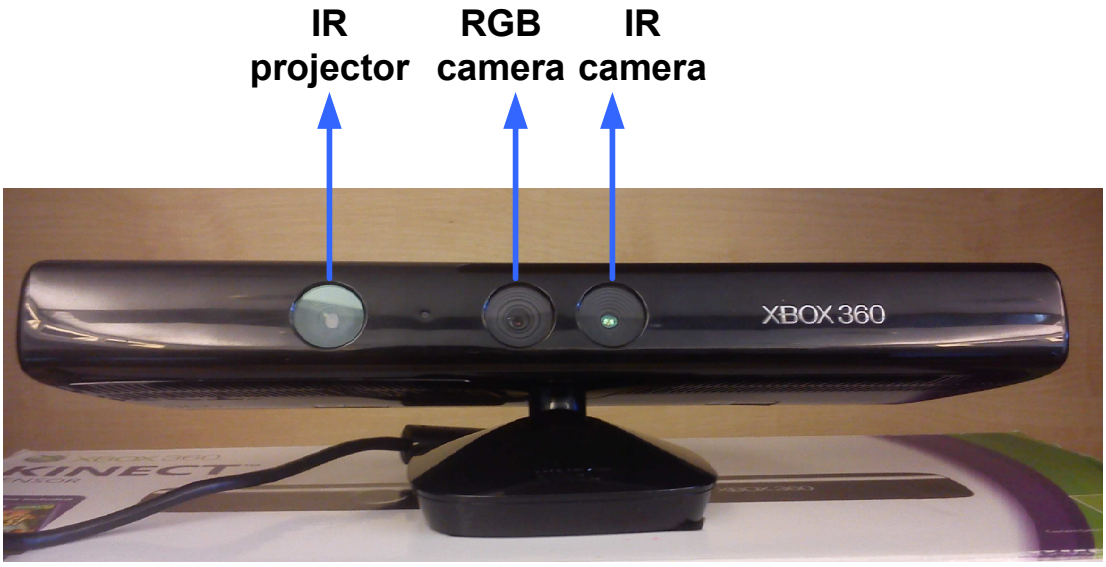


Figure 2.2: RGB and depth sensors in Kinect

The IR projector sends a grid of IR beams, a method known as "structured light". The sensor is calibrated so that it can identify the depth of objects using the displacement of each beam projected from it in the reflection received by the sensor. This process of creating a depth image is handled by custom hardware from a company named PrimeSense[2]. Sensing the colour and depth of a scene concurrently allows extracting the 3D structure of the surroundings, thus creating a representation of the environment.

A third type of sensor available in Kinect is an array of 4 microphones, placed

[2] http://www.primesense.com/

so that captured sounds can be located within the room. This feature allows the sensor to distinguish voice commands if it is being used by multiple users.

One of the interesting features of Kinect is that it can also track human skeletons, based on the approach presented in [36]. Kinect achieves this by first creating a depth image and then identifying 31 different parts of the body (the joints). The second stage infers the body pose using a decision forest, a machine learning method making use of the results from multiple decision trees, trained with a large set of training samples ($500,000$ frames with ground truth captured using a motion capture system).

Kinect's RGB camera can be calibrated using the Kinect Calibration Toolbox [37] or the calibration parameters can be obtained using another method [38] that is similar to stereo calibration [39].

Since its launch, Kinect has attracted much attention from both game developers and vision researchers; indeed, some artists create artwork using it. A range of different applications making use of Kinect is given in [40].

Kinect is used in this thesis for finding planar features in the environment for *in situ* augmentation and to detect human users for augmenting them with ancient clothing. Both of these are presented in chapters 5 and 7.

2.1.2 Feature detection and description

A *feature* is an image primitive that contains valuable information about the content of the image. Every feature appearing in an image shadows a real-world object. A feature can be in form of a corner [41], an edge [42], a small region (blob) [43] or a segment [44].

Features are represented using *descriptors*, which are calculated using the pixel

information around the feature using a variety of methods: A small patch of surrounding pixels can comprise the descriptor, or a more complex description like an oriented gradient histogram [45].

The literature presents many different feature detectors and descriptors. An evaluation of many feature detectors can be found in [46]. Based on the review given therein, a good feature detector should be able to detect features that are stable in terms of geometry under different viewing conditions [47, 48], should present significant amount of variation in its neighbourhood so that they will be prominent and provide useful information as well as presenting good localization accuracy [49]. It is also important for the detector to detect such features in a reasonable amount of time, a vital requirement for real-time applications. The feature detectors and descriptors used in this thesis are briefly outlined in the following paragraphs.

Harris & Stephens [50] is a feature detector that uses eigenvalues of the Harris matrix as a measure of an image feature's prominence according to the values in the local neighbourhood. It is reported to extract features that are invariant to changes in rotation and illumination [41, 47].

Based on the research performed by Tomasi and Kanade [51] in order to specify a formulation for image displacement (*i.e.* tracking the features in the image), the Kanade-Lucas-Tomasi (KLT) operator was derived. Similar to the Harris & Stephens operator, this detector also works on eigenvalues but the criterion for selecting corners is that the first eigenvalue should be larger than a threshold. A further improvement was proposed in [52], suggesting that the smaller of the two eigenvalues should be larger than a predefined threshold value to make a good feature. This detector is also known as the Good Features to Track (GFT) operator.

The Smallest Univalue Segment Assimilating Nucleus (SUSAN) [49] operator analyses a pixel point by placing a circular template and terms this the 'nucleus'. The number of pixels having a similar brightness (which defines the Univalue Segment Assimilating Nucleus (USAN)) to the nucleus is counted and subtracted from a geometric threshold in order to create a response image. A test for false positives (*i.e.* a feature that is not actually a corner but reported as a corner by the detector) is then applied using the centroid and contiguity of the USAN. Finally, non-maximum suppression is applied.

Edge-based Regions (EBR) [53] defines affine regions by making use of the neighbouring edges around a corner point. The corner point is associated with straight or curved edges to define the affine invariant parameters which are robust against changes in affine geometry (rotations and translations) as well as photometric variations such as changes in lighting conditions.

Intensity-Extrema-based Regions (IBR) [54] first finds local extrema based on intensity. The next step is to cast rays from each extremum to its surrounding and creating an intensity profile for these rays. The point where the intensity changes significantly is signed with a mark and then an ellipse is fitted to find the region limited by the marks.

Features from Accelerated Segment Test (FAST) [55] finds interest points by testing each pixel using a Bresenham circle of radius 3 (16 pixels in total). If N of these contiguous pixels are similar *i.e.* have an intensity value close to the centre within limits, then it is selected as an interest point. Note that values between 9 and 12 can be used as variants for N — though too many interest points are found when $N < 12$ [56]. To improve the performance (Features from Accelerated Segment Test-Enhanced Repeatibility (FAST-ER) [56]), the values on the circle are compared with the centre point to see if they are darker, similar or brighter.

Then, a decision tree can be trained to compare the 16 pixels against these classes and decide whether the centre is an interest point or not. Again, non-maximal suppression is used to eliminate clustered interest points.

Scale Invariant Feature Transform (SIFT) [57] works by selecting candidate key-points from locations which can be repeatedly chosen under different orientations and scales. Scale invariance is achieved by using a "scale space" which appears as a pyramid of images consisting of the octaves created by resizing the original image to its half size and then applying a Gaussian blur operation. Key-points are found using a method called Difference of Gaussians (DoG) as an approximation of Laplacian of Gaussian (LoG). A local descriptor is then generated by calculating the magnitude and orientation of the gradient. Later, a feature vector is computed using a histogram of these orientations.

Speeded-Up Robust Features (SURF) [58] were developed as an improvement to SIFT for extracting features in a shorter time, employing integral images as an intermediate image representation and using Hessian-Laplacian to approximate LoG. For the description, Haar wavelet responses inside a circular window are summed to obtain the orientation vector of the feature. SURF is also claimed to be more invariant to affine transformations such as translations or rotations than SIFT by its authors.

It is known that scale invariance on its own is not enough to show robustness against changes in viewpoint, which result in affine transformations in the image [59]. For this reason, a number of affine-invariant feature detectors have been proposed. Harris-Affine (HarAff), Hessian-Affine (HesAff) combine scale space with Harris and Hessian-based detector are detectors developed for additional robustness against changes in viewpoint. Similarly, LoG is combined with Harris and Hessian-based detectors again in order to achieve invariance against scale in

Harris-Laplace (HarLap), Hessian-Laplace (HesLap) [59]

Scale invariant Feature OPerator (SFOP) [60] is a scale-invariant feature detector that can find invariant features from corners, junctions and features that have a circular form by combining two existing approaches (junction type points [61] and circles using a spiral model [62]) into a single framework. The detector can provide good complementarity [63] to other feature detectors since it can find features that are not detected by others.

Binary Robust Independent Elementary Features (BRIEF) [64] perform a simple comparison between intensity values of an image patch and its smoothed version. In this way, a binary descriptor that can have different lengths (*e.g.* 128, 512) is calculated. The binary nature of the descriptor allows matching using the Hamming distance metric, which can be implemented efficiently using an Exclusive OR (XOR) operation.

Oriented BRIEF (ORB) [65] is a detector/descriptor which uses the FAST detector for finding features and BRIEF to extract the descriptor. The FAST detector was made orientation-aware by the addition of an orientation metric known as intensity centroid. BRIEF was also enhanced by employing a matrix which is rotated by angle increments for the binary tests, making the descriptor rotation invariant.

As mentioned earlier in the section, apart from engineered descriptors like SIFT or SURF, a template can be used to describe a feature. A template or a patch is a region of pixels extracted from the neighbourhood (*e.g.* 9×9, 11×11, *etc.*) of a feature and can be seen as the most basic method of describing a feature (Figure 2.3). Increased patch size adds to the computation time and patches of size 15×15 are reported to provide sufficiently stable features for long term processing [66, 67].

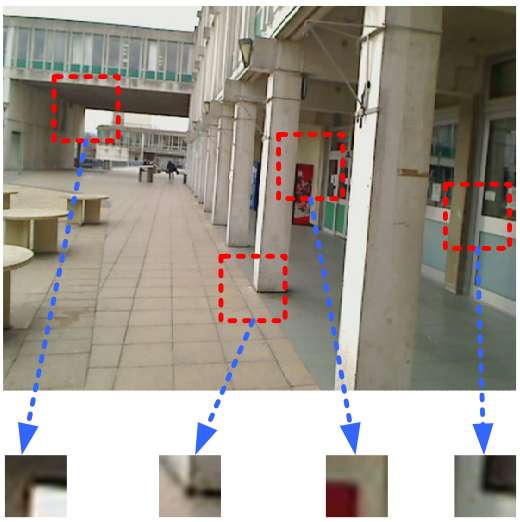


Figure 2.3: Templates extracted from different manually selected features

Feature descriptors are used to match images to find correspondences between them which will later be used to understand the spatial change in feature positions, and hence the camera motion as described in section 2.1.3. Template matching involves finding a given patch inside an image. Normalized Cross-Correlation (NCC) is the method usually employed for template matching [68], though other methods are also available (*e.g.* Sum of Absolute Differences (SAD) or Sum of Squared Differences (SSD)). The response of NCC will be a peak with value of $\simeq 1.0$ where the template is found.

Template matching can be a good initial approach for description when the features are detected with the GFT method or the FAST detector. When a feature is detected, an image patch is extracted as the signature of that feature. If this feature is visible in the current frame, it is searched within the image. The result of the NCC is then checked against a threshold (*e.g.* 0.7) for a successful match. The search for a template can be optimized by providing an initial estimate of the template position [68] which can be the predicted 2D location of a feature from a

filter, described in chapter 4 [69].

Comparison between two descriptors is conventionally made using the Euclidean distance and the feature producing the best result (*i.e.* smallest descriptor distance) is selected as the matching feature, the Nearest Neighbour (NN). When matching is performed using the Euclidean distance, the search process for matching features can also be optimized using the search region to define possible match candidates, as mentioned earlier. For increased robustness, an additional criterion can be added such as a match being only accepted if the distance of the NN is significantly lower than that of the second NN [57].

More recently, the Fast Library for Approximate Nearest Neighbours (FLANN) method [70] has become popular for feature matching for SIFT and SURF descriptors. FLANN makes use of either a randomized kd-tree or a hierarchical k-means tree, depending on the feature dataset, to optimize performance for speed.

Once a set of matching points are found, these correspondences must be checked for incorrect matches. RANdom SAmple Consensus (RANSAC) [71] is a method which is used to select the best model for the image transformation based on a minimum number of matching points between the source and destination images. The method takes a random set of samples and applies the transformation, then calculates the number of samples that are compatible with this transformation 'inliers'. This number serves as a support for the selected model and after several iterations the model with the highest support is selected as the consensus. Incompatible matches ('outliers') indicate incorrect matches. A comparison of RANSAC derivatives is presented by Cho *et al.* in [72] and more recent derivatives are also available (*e.g.* BetaSAC [73]), claiming significant speed-up over RANSAC.

A dozen feature detectors are evaluated according to a novel criterion presented

in chapter 3 and some of these are used in the vision-based tracking algorithm of chapter 4 and the *in situ* augmentation algorithm of chapter 5.

2.1.3 Structure from motion

Structure From Motion (SFM) aims at finding the camera location with respect to the environment and constructing a map of the environment. Finding the location of the camera is termed the 'absolute orientation problem' and involves the estimation of the rotation matrix R and the translation matrix t [74]. A typical SFM approach follows the steps given below [75]:

1. Robustly detecting of salient features, points or lines
2. Determining the correspondences between these features
3. Updating the scene structure estimation

Methods for the first step have been described in section 2.1.2. The second step has been studied for a long time and a number of approaches can be found [76–78]. A common point in many algorithms is that the camera pose is obtained by finding correspondences between the 3D coordinates of features and their 2D projections in images, a problem known as Perspective-n-Points (PnP) [78]. For three correspondences, four possible solutions can be found, whereas a unique solution can be found for six or more correspondences [79].

Finally in the third step, the estimate of the camera pose and the scene structure is updated using the updated information from these 3D–2D correspondences. An optional refinement stage after the third step can be done either using a filtering approach (section 2.2) or a more expensive method known as *bundle adjustment*.

Bundle adjustment [80] is an iterative method for refining the camera pose estimates and 3D point coordinates using an optimization technique (usually Levenberg-Marquardt [81]). The aim is to minimize the projection errors for the 3D points using the camera parameters that model its position and orientation. This approach is computationally expensive and, even for a small number of camera parameters, may take hours to converge. Sparse versions of the algorithm, aimed at improved efficiency, are also available [82].

Lu *et al.* [83] devised an algorithm which computes the rotation and translation matrices from object reference frame to camera reference frame using an orthogonal iteration algorithm based on Singular Value Decomposition (SVD). The algorithm requires camera intrinsics and can also improve upon the initial estimate. One problem with this method is that it could result in pose ambiguity for planar targets as indicated by [67, 84]. To address pose jump problems due to two local minima of the error minimization function for pose estimation, Schweighofer *et al.* [84] improved the SVD solution in [83] for planar targets where they used 4 coplanar (but not collinear) interest points.

Motion estimation of the camera and acquisition of the 3D structure of the environment was achieved by tracking and matching Harris points between frames acquired at video rates with a fast local bundle adjustment technique in [85]. This approach was applied each time a new camera pose was added to the system. First, a triplet of images was captured to initialize the global frame and geometry; then some frames were considered as keyframes for the triangulation of 3D points. Robust pose calculation was performed for each of the captured frames. Keyframes were added to the system when the number of matching points with the last keyframe fell below a threshold or if the uncertainty of a calculated position was high.

Chandraker *et al.* [74] applied vision-based localization in a room using a novel target design having exactly four corners along each straight edge. One disadvantage of this system was the need for re-initialization each time the camera lost its track.

The system in [86] started from an initial estimate of the state vector; from that, the developed method is able to determine the sensor position and the structure of the environment for mobile AR, combining visual and inertial data using an off-line learning process for object recognition in which the system was trained with a set of interest points. Developed approach is used in an extended tracking operation where these points disappear and new points appear in the frame. The authors reported about the need for a multiple model system as in [87].

A recent PnP solution known as Efficient Perspective-n-Points (E-PnP) is presented in [88,89]. The method selects 4 virtual non-coplanar control points from a set of 3D reference points to generate and solve a linear system of equations. The selection of the control points is based on the principal axes around the centroid of these reference points. This approach can handle planar and non-planar cases well while scaling linearly with the number of 3D–2D correspondences.

SFM methods generally involve off-line batch processing, do not guarantee repeated localization [86, 90] and provide limited robustness [75], whereas on-line pose tracking is required for AR applications. Noise can also become a problem, especially in PnP approaches [78]. Furthermore, pure SFM methods present high computation costs. Due to these problems sensor fusion was suggested in different studies [74, 91].

The E-PnP algorithm is used along with the Kinect sensor in chapter 5 in order to find the camera pose using the 3D points from the depth sensor and their

2D projections in the RGB image.

2.2 Filtering Methods

In the state estimation problem, where a physical phenomenon is modelled using dynamic variables, one needs to observe the phenomenon in order to take measurements and hence update the model. This process involves handling erroneous and incomplete information from the model itself and the tools used for making observations. "Filtering" methods are used in these sorts of cases. Note that filtering here is not only used for noise removal but also for incorporating the measurements in the state estimation with an ultimate aim of obtaining the best estimate of the state from data that are corrupted with noise.

Filtering methods can be classified into two broad groups, namely Gaussian techniques (*e.g.* Kalman filters and its derivatives) and nonparametric filters (*e.g.* particle filters). Examples of how filters from both types are used in this thesis will be given in chapter 4; however detailed information and discussion on both types can be found in [92]. The following subsections will briefly summarize the filters used in the thesis.

2.2.1 Kalman filtering

The Kalman Filter (KF) [93] can be considered a practical implementation of a Bayesian filter which works by updating a prior probability from some testing evidence (measurements) in order to obtain the posterior probability. These probabilities are actually the state of the system, as mentioned above.

The KF assumes that the system is linear and affected by Gaussian noise, and models the state using a uni-modal distribution represented with a mean and a

covariance (moments parameterization) [94]. Under these two conditions, the KF is considered to be optimal [95] in the least square sense, meaning that the mean square error of the estimated state parameters are minimized by the filter.

A KF has three main processing stages namely *prediction*, *measurement* and *update*. In the first stage, the filter generates an estimate using a transition function. Next, measurements of the physical phenomena are taken. In the last stage, these measurements are used to update the filter, preparing it for the next iteration.

Derivatives of the KF also follow the same stages. One such popular derivative is the Extended Kalman Filter (EKF) which is aimed for handling non-linear cases. EKF performs this by using a Taylor expansion to linearise the system dynamics and the observations for the filter.

The KF and its derivatives have been applied to a wide range of applications from target tracking [96, 97], sensor fusion and robot navigation [98] to machine learning (*e.g.* training neural networks [99]).

In the thesis, KF was used to integrate GPS, IMU and camera measurements to track a walking user in an outdoor environment.

2.2.2 Particle filtering

A Particle Filter (PF) is essentially a Monte Carlo approach [100] which models the uncertainty of the state estimation problem, starting with a random distribution over the state space and then converging to a denser solution depending on the measurements. Unlike the KF, a multi-modal distribution is modelled in the PF *i.e.* the posterior belief of the system state is estimated using a finite number of samples ('particles') that populate and ideally cover the entire state space.

Each particle stores a hypothesis of the state. Practical applications use a high number of particles in order to cover the each possible value of the state [94]. A PF replaces the update part of the KF with a process called re-sampling in which weights of particles, indicators of confidence for a particle, are updated based on the measurements. After this sampling process, a new set of particles are drawn from the initial distribution based on the weights for the next iteration.

A large number of particles is required to cover the complete search space, otherwise the correct state may be missed, a problem known as particle deprivation [101]. The actual number of particles may be selected based on the size of the search space. A trade-off comes into play as each particle added to the system and can significantly increase the computational cost.

Due to its inherent ability to model multiple hypotheses, PFs have been used in many applications, from computer vision to robot navigation. A popular application in computer vision is the *Condensation* algorithm [102] which is used for visual tracking. In robot navigation, Montemerlo *et al.* [103] developed an algorithm for a mobile robot to maintain its position based on the readings acquired by its sensor.

In this thesis, a PF was used to track centres of rectangular objects for the *in situ* augmentation algorithm presented in chapter 5.

2.3 Simultaneous Localization and Mapping

SLAM (*a.k.a.* Concurrent Localization and Mapping (CML)) is the dual of SFM in the robotics community and is defined as follows: When a robot is put in an unknown environment, it has to create a map of the environment and must localize itself based on this map at the same time, as shown in Figure 2.4. A 'map' here is a

spatial representation of the environment, simply a list of objects stored with their positions [94] as the robots cannot currently interpret the conventional maps that humans use. However, the primary difference between SFM and SLAM is that SLAM methods usually involve a recursive estimation for real-time autonomous operation whereas SFM methods present batch solutions [104].

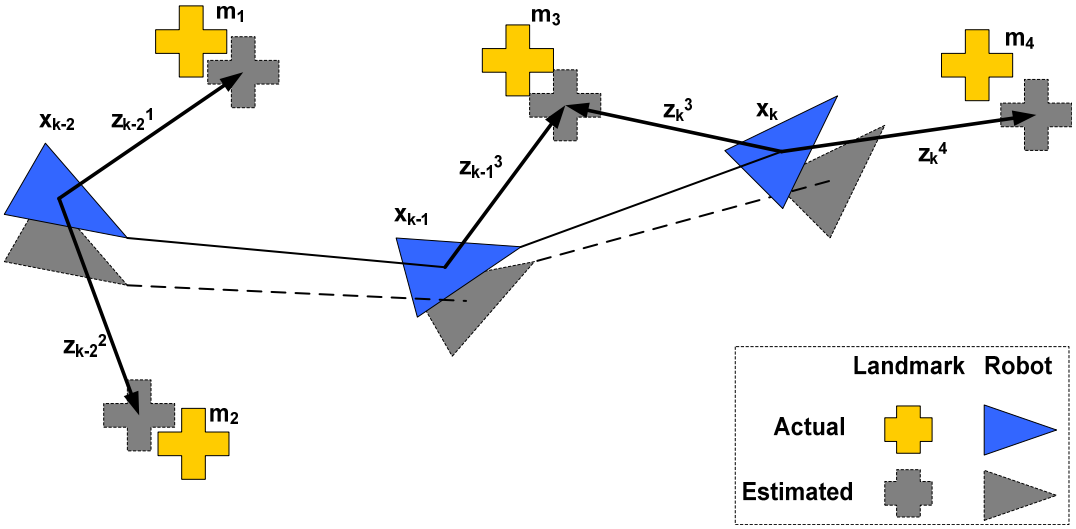


Figure 2.4: SLAM problem. A filtering mechanism is used to estimate robot's position x_k at time step k. In order to update this filter, the robot needs to take measurements ${z_k}^i$ from the surrounding landmarks m_i where $i \in (1, 2, 3, 4)$. It is assumed that the sensors of the robot are noisy so the robot will actually be measuring a distorted image of a landmark rather than the landmark itself. Following [105].

SLAM has been used in robotics applications [106] both indoors such as robotic arms, wheeled-robots, walking robots; and outdoors such as autonomous vehicles, underwater applications for submarines or sponge bed surveys, UAVs or planetary rovers. Apart from robotics, SLAM has also been used to find contour clusters belonging to an object by considering the object's skeleton as an imaginary robot's trajectory [107].

2.3.1 SLAM in general

A general review of how SLAM methods have evolved was presented as a two–part tutorial in [105, 108]. The first part of the tutorial explained the foundations of the problem and stated the two most common methods namely, EKF and PF techniques [105]. More advanced concepts such as methods to reduce the computational complexity or data association issues about SLAM were presented in the second part of the tutorial [108]. The core problems of SLAM are stated as the data association problem [109, 110] and computational complexity [111, 112].

Data association

Data association [94] concerns finding a unique correspondence between the representation of a feature and the real-world object from which the feature is extracted. It is the process of extracting a unique relation between the sensor measurements and the objects in the map. Therefore, if the measurement produces a spurious pairing (*i.e.* a false match), this will badly affect the output of the localization algorithm.

Neira *et al.* [109, 113] showed that the classical NN algorithm for data association in stochastic mapping methods such as the EKF framework was sensitive to the increase in vehicle or sensor error. A Joint Compatibility Branch and Bound (JCBB) algorithm was devised to obtain more robust estimation and prevent false matches.

The importance of data association arises once more for the problem of loop-closing, which can be described as recognizing a previously visited location. Stability and robustness of features play an important role in this [114], as the map will grow infinitely and the accuracy of localization will diminish when loop-closing

can not be achieved. Loop-closing in [115] was achieved for several hundreds of meters using the JCBB test for robust feature handling for urban areas. Two important problems were reported as moving objects, which could provide good matches but did not act as static landmarks for repeatable localization; and ambiguities in the images such as repeated textures *e.g.* grass or tiles. RANSAC was indicated as an alternative method [115].

Computational complexity

Computational complexity is an important concern in SLAM systems and depends both on the map size and the performance of data association. A larger environment needs more features to be added to the map, while the increase in map size will reduce the performance of the system.

Hierarchical mapping (*a.k.a.* sub-mapping) approaches are used to prevent this problem by storing the whole map in form of smaller maps and performing localization using these smaller maps [108]. The advantages of this approach are identified as:

1. The number of features to deal with will be limited. This will reduce the computation time required to update a map [116].

2. Uncertainty in a local map will be smaller than in a bigger map. This will increase the accuracy of the system [110].

Bailey [110] used Network Coupled Feature Maps (NCFM) to solve the problems of consistency and tractability of large scale SLAM systems. Similarly, Claracco [117] investigated extending SLAM to large-scale environments using topological map building. Another important consideration in dealing with the

sub-maps is also related to data association since it will be used in overlapping areas to join two maps into a larger map [116].

2.3.2 Visual SLAM

Visual SLAM is a sub-branch of SLAM for the specific case where vision is used. There are several reasons for this [118, 119]. First of all, cameras are cheap, small in size and light. Second and more importantly, robust features can be extracted using computer vision. Third, visual motion estimation techniques allow 3D SLAM and finally, accurate motion estimates can be obtained using these techniques.

Based on the number of cameras, we can classify visual SLAM systems into two broad categories: monocular and stereo. Lemaire *et al.* [119] explained visual SLAM for both categories; The methods do not differ much. When stereo vision is used, the baseline between the cameras is used to calculate the distance to the feature observed by both cameras. When there is a single camera, this baseline is obtained by moving the camera and observing the same feature from different locations. Camera calibration can be used for both approaches.

Methods used for visual SLAM, which can also be applied to other sensors such as laser range-finders or sonar, can be classified into three groups according to the algorithms used. The first group is the KF approaches, which use a recursive prediction–measurement–update sequence to estimate a robot's position [120]. The second group is the PF approaches which have been proposed as alternatives to the KF algorithms [94]. The difference between these approaches is that the former stores the estimate as a single state where the latter uses a set of particles to store several estimates. The third group of approaches either combines KF

and PF in order to achieve reliable mapping [121] or uses bundle adjustment to perform frame-to-frame matching [122]. The literature presents a vast amount of applications from all three groups: for space considerations the discussion will be limited to applications in AR in section 2.7.1.

This research initially investigated user tracking with purely visual SLAM; however due to the problems discussed in chapter 4 the approach was adapted to a combination of SLAM and SFM.

2.4 Fuzzy Logic

The main idea behind fuzzy logic is that the truth value of a proposition is represented by a function which maps propositions with an infinite set of numbers, or in numerical terms the interval $[0, 1]$, instead of the conventional binary logic (*i.e.* true or false). This idea originates in Zadeh's initial work on fuzzy sets [123] where membership of an element to a set is regarded as a partial membership rather than black and white decisions such as "is/is not a member". Having multiple truth values provide a good representation framework for handling uncertainties and this representation has a very significant value that allows reasoning in an approximate way [124].

Fuzzy logic can be used to find an approximate model for a problem and solve it in a way which is not mathematically complicated. This is achieved by using linguistic variables which are normally used to represent verbal uncertainties. For instance, when humans talk about the speed of a car they can either use a direct measure like 120 kilometres per hour in a crisp form or use words like "fast" or "very fast". The second form of expression allows creation of verbal rules in the following form:

"*IF* the speed is slow *THEN* accelerate."

or

"*IF* the speed is very fast *THEN* slow down."

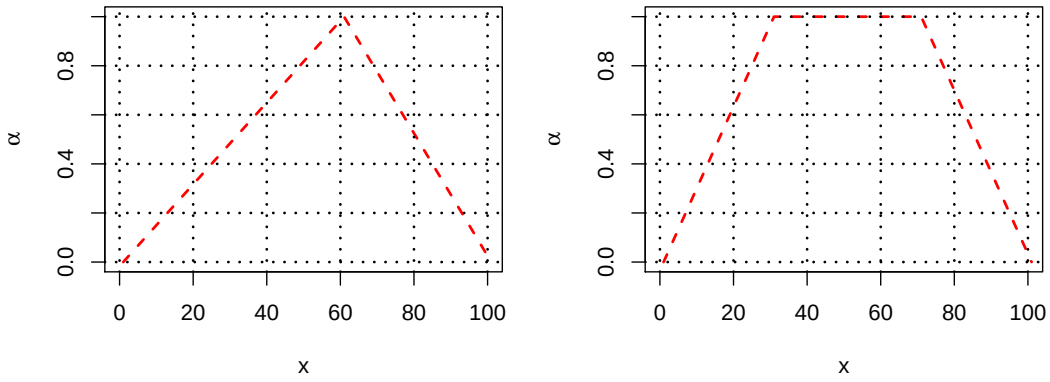
A set of similar rules, stored in a rule-base, can be used to define the behaviour of a system. Membership of a crisp input value to these linguistic variable can be tested using membership functions. The most commonly used membership functions are the triangular, trapezoidal, Gaussian and bell-shaped functions shown in Figure 2.5.

Considering a system that automatically adjusts the speed of a car following the example given above, a membership function can be defined as in Figure 2.6 to map different linguistic variables given the crisp speed of a car.

The process of converting a crisp value to a linguistic variable using an input membership function is called fuzzification. For the car example, once the car's speed is fuzzified and expressed as a linguistic variable then a rule can be selected to decide the action that will be taken based on the car's speed.

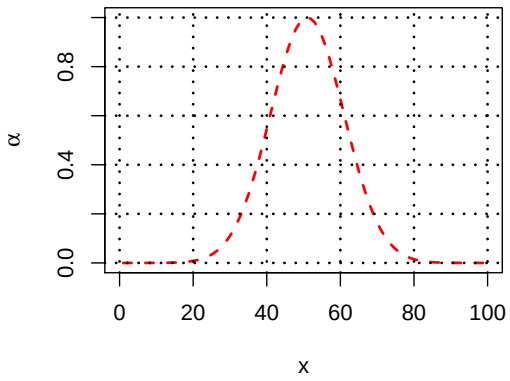
From the example rule, it can be seen that the action is to slow down if the car's speed is very fast. Humans can easily interpret such a command but obviously a computer system needs to know the target speed and the deceleration to be applied to actually slow down. This can be done using a conversion from the linguistic variables to crisp values using an operation called defuzzification, the opposite of fuzzification.

Using the processes mentioned above (*i.e.* creating a rule-base, fuzzification and defuzzification), a Fuzzy Logic Controller (FLC) [125] can be created to control the behaviour of a system. A sample digram is shown in Figure 2.7.

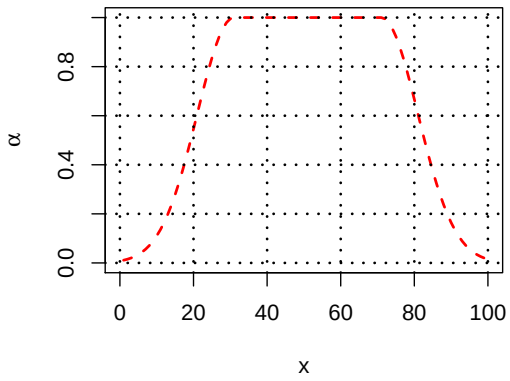


(a) Triangle

(b) Trapezoid



(c) Gaussian



(d) Bell-shaped

Figure 2.5: Membership functions

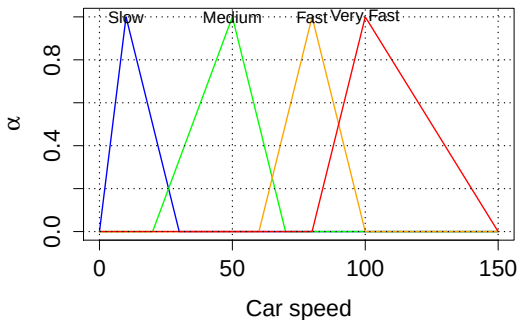


Figure 2.6: Membership functions for a car's speed

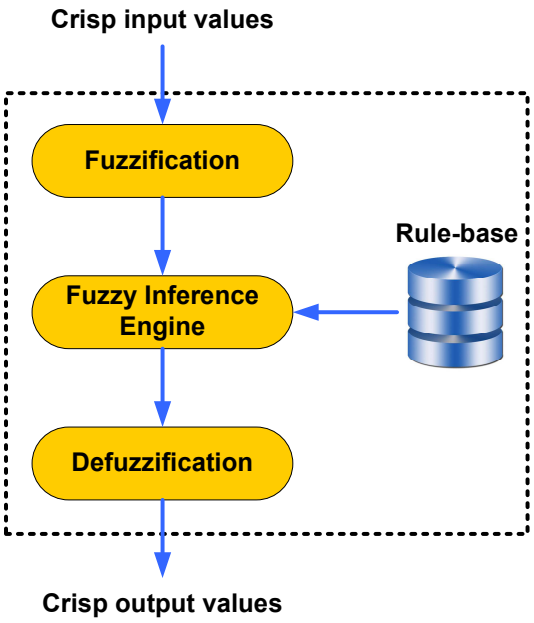


Figure 2.7: Diagram of a fuzzy logic controller. Following [125].

A problem that can be encountered when designing a system using fuzzy logic is the exponential growth of the rules if the numbers of input and linguistic variables are large [126]. Rule reduction methods are available to overcome this [127, 128].

With their inherent ability to handle uncertainties and deal with complex non-linear systems, FLCs have been used in many applications, such as manufacturing [129], automated control [130] and many other real life problems [126]. Recently, Hacıomeroğlu *et al.* used fuzzy logic to control movements of individuals for simulating human behaviours in pedestrian groups realistically [131].

In this thesis, fuzzy logic was used to define behaviours for using multiple motion models in order to apply a better model that can fit the user's motion in chapter 6.

2.5 Statistical Analysis

Statistical analysis is a quantitative approach for describing a population which consists of individuals. Statistical analysis can be performed on samples (*i.e.* a representative subset of the population) of a population for two purposes [132]:

1. Extract clear, concise and accurate summary information from samples using descriptive statistics;

2. Make predictions on the population by first creating a mathematical representation of the population and extracting/guessing relations.

The second purpose is a specific type of statistical analysis since it involves testing a hypothesis that was put forward with a hope that it may provide an understanding for an observation [133]. This initial hypothesis is called the *null hypothesis* shown with H_0 and suggests that the means from different populations are equal. The null hypothesis is supposed to be true by default though it needs to be tested. There is also an *alternative hypothesis* (H_1) which is competing with the null hypothesis and suggesting that at least one of the means is different.

An example for hypothesis testing is analysing the performances of a group of workers working in a production line. The null hypothesis is that the performances (*e.g.* number of items produced in a day) of all workers are the same, whereas the alternative hypothesis is that there are significant differences between their performances.

Data about the numbers of items produced by each worker could be collected over the course of two months and used to test the null hypothesis. Results could reveal that the workers produce different number of items; if so, the null hypothesis must be rejected in favour of the alternative.

The literature presents methods to perform hypothesis testing such as t-test or z-test for comparing means of two populations (when sample size $n < 30$ or $n \geq 30$ respectively) and ANalysis Of VAriance (ANOVA) for comparing two or more populations [134].

Before examining ANOVA in more detail, it is worth mentioning that two types of errors can be faced when performing hypothesis testing, known as Type I and Type II errors. The former type of error occurs when the null hypothesis is rejected even though it is true, whereas the latter arises when the null hypothesis is not rejected when it is actually false.

2.5.1 Variance analysis

As a generalization of t-test for more than two samples, ANOVA performs hypothesis testing by comparing the means of several groups to see whether or not they are equal. There are two reasons for using ANOVA instead of running t-tests between pairs of groups: One reason is that pairwise comparisons between samples is time consuming when there are more than two samples. Second and more importantly, one can expect to have one of twenty t-tests performed to be incorrect since this is a consequence of the 5% acceptance level.

ANOVA assumes that independent samples are drawn from populations which follow a Normal distribution and have equal variances. However, ANOVA is also known to be robust where the homogeneity of variances assumption is not satisfied [135]. A transformation (*e.g.* log transformation [136]) can be used to satisfy this assumption since it results in an equal spread.

The result of the transformation can be tested using Hartley's F_{max} test [137], which involves finding the ratio of the largest variance to the smallest (to yield

F_{max}) and then looking-up a table of critical values [138] for the given number of treatments and degrees of freedom. If F_{max} is found to be smaller than the value in the table then it is safe to assume the homogeneity of variances assumption.

ANOVA uses a measure known as the F statistic, the ratio of the sum of squares between the groups and sum of the squares within the groups, after both have been divided by their number of degrees of freedom. This calculated statistic is then compared against a threshold value, $F_{critical}$, which is obtained from an F ratio table for the number of degrees of freedom. If $F > F_{critical}$ then the differences between the groups are statistically significant and H_0 can be rejected.

Finding whether a difference exists between different populations is certainly important; however in most cases it is more informative to perform pairwise comparisons between means and ascertain a ranking between these populations (*e.g.* which worker is performing better).

2.5.2 Multiple range test

Duncan's *multiple range test* [139] is a test that is used to compare means. It makes sense to use such a test when the null hypothesis is rejected after the ANOVA test to see the actual differences between the means.

The test works by first sorting the means of the populations into descending order. Then, the highest mean is compared with the lowest one. If the difference between the two is greater than the residual mean square, calculated using a table called "Studentized range" or Q table [138], again for a given number of degrees of freedom and number of treatments (populations), then it is concluded that the difference between them two is statistically significant. The test is continued by selecting the second highest mean and comparing with the lowest mean. Each

time the number of treatments is reduced by one before consulting the Q table.

An alternative method is Fisher's Least Significant Difference (LSD) [140] which is known to be more sensitive to Type-I errors and less discriminating than the multiple range test [141].

Chapter 3 presents an evaluation of feature detectors using the statistical analysis methods described in this section according to a novel criterion, feature coverage, which is shown to affect homography estimation.

2.6 Augmented Reality

AR is the process of blending real-world images with artificial objects or information generated by the computer. It is defined as an extension of user's environment with synthetic content [142]. AR can also be used to enrich human perception and facilitate the understanding of complex 3D scenarios [143, 144].

A broad and comprehensive survey of AR was given by Azuma [143]. Although his review is not recent, the topics he defined in those earlier days of AR still compose the broad categories of today's applications. These categories were defined as medical, manufacturing and repair, annotation and visualization, robot path planning, entertainment and military training.

The literature presents a wide variety of applications, including 2D or 3D, augmentations on different platforms. For 2D applications, AR with an overlay of information about buildings as annotations on images was presented in [145–147]. Several examples can be found for 3D augmentations [148–152].

In the study given in [153], four prototype interfaces were presented for mobile devices from which users can make queries for the nearest market or pharmacy including list, map, Virtual Reality (VR) and AR interfaces. Query results were

presented geographically based on the user's current location in the list interface and displayed on a backdrop map in the map interface. The VR interface was presented as an alternative to 2D map, providing a 3D view of the location. Finally, the AR interface was used to merge geographical information with the real-world scene.

AR was used in human way-finding tasks for military search and rescue operations in [154]. Two types of users were put in a maze of size 4m×5m. The first type of users were given a map and told to memorize it before entering the maze. The second type of users were aided with a wearable AR system displaying their location in the map. The results showed that augmented users spent less time to localize themselves in the maze.

The AR system in [152] aimed to provide field workers of utility companies such as electricity or telecommunications with an enhanced perception for outdoor tasks, including maintenance of underground infrastructure using a hand-held device. The system used a Geographical Information System (GIS) database for displaying 3D augmentations of hidden structures such as cables or tubes over their actual positions. Another mobile GIS system was presented in [155] with the use-cases given as outdoor localization and guidance, military tactical planning or urban planning.

Interactive 3D designer applications for indoor [156] and outdoor environments [151] were presented following the studies of Piekarski [157], who used Constructive Solid Geometry (CSG) in order to create 3D constructions. Depth estimation was performed using AR working planes which were generated relative to other objects in the environment.

Following the ARQuake developed using the system in [148], different mobile AR games were presented such as Human Pacman, NetAttack, ARSoccer, ARTen-

nis, The Invisible Train and Capture the Flag in [142]. Furthermore, audio-based AR is also well known *e.g.* the mobile narrative-based audio game in [158], though AR is generally associated with imagery and vision. This idea can also be used for other AR applications and Human Computer Interaction (HCI) studies considering visually impaired people [159].

The evaluation of AR systems is an important topic as any developed system is supposed to serve its purpose well. Different evaluation methods, such as heuristic, formative or questionnaire-based, are discussed in detail in [160]. A user-centred evaluation of the CONNECT AR system was presented in [161] where AR was viewed as a learning method for science and history museums. The evaluation criteria included visual discomfort, dryness or irritation in eyes, difficulty in focusing, visual fatigue, dizziness, nausea and general tiredness. Another evaluation was presented in [149] as a questionnaire-based user assessment to investigate how users felt during and after using a wearable system.

2.6.1 Cultural heritage applications

Cultural heritage applications [162] such as the reconstruction of ancient Olympia in Greece [163, 164], Paestum in Italy [165] or the Gosbecks Archaeological site in Colchester [149] were added to the list of application types given in [143] by Papagiannakis *et al.* [166] . AR is an interesting topic, especially for archaeology, since it is a good approach to displaying ancient buildings to tourists in the same form as they were in the past. This usage has another valid reason, which is the opposition of archaeologists to physical reconstruction *in situ* as they prefer keeping the original structures for future generations [163]. By using an AR system, no physical reconstruction is required but the computer-generated reconstructions

can be overlaid on the ruins in the site.

An alternative method for this task is VR, which allows simple tours among several buildings [167] or with user interaction. Laycock *et al.* [168] presented an interesting application that allows users to view high-fidelity reconstructions of ancient buildings and how they changed (*e.g.* partial demolitions or extensions to the buildings) over time.

Ryder *et al.* [169] worked on populating cultural heritage reconstructions in order to improve the presentation of the models by adding a lively scene. Methods for optimization were shown in order to avoid performance problems due to rendering the large amount of detail introduced when many human models are added to the scene. This idea is similar to the dream that motivated this research, mentioned in chapter 1; however these models are synthetically generated rather than augmented images of the users.

The ArchaeoGuide project [164] used Virtual Reality Modelling Language (VRML) models, images and sounds for display. Users could customize the display according to their needs. This system was reported to be uncomfortable to wear, fragile for outdoor environments and to have contrast problems in sunny days. In [165], the actual position of the user was used to determine the position of an avatar in the virtual world for the archaeological site. The system was not a real AR example but users could see the virtual city as they walked amongst the ruins.

The Kinect has also been used in the context of cultural heritage. For instance, Remondino [170] presented a review of using different types of imaging and depth sensors, including Kinect, to perform 3D scanning of archaeological objects for the purposes of digital recording, historical documentation and preservation of cultural heritage. Richards-Rissetto *et al.* [171] used the Kinect's body motion

detection features to perform navigation in a 3D reconstructed model of an ancient Mayan city using virtual reality.

The thesis presents several AR applications for cultural heritage (displaying State *Agora*, a meeting place for government and business discussions in ancient Greece; augmenting synthetic columns over rectangular objects; and a simple game) in chapter 7.

2.7 User Tracking Methods for Augmented Reality

Tracking, the process of locating a user in an environment, is critical to the accuracy of AR applications as more realistic results are obtained in the presence of accurate AR registration. This process includes determining the position and orientation of the AR user.

Generally, the most important part is tracking the head, as the user typically wears a HMD on which the augmented images of the real world are displayed. Furthermore, a tracking system mounted on the head has better signal reception if GPS will be used, has a good Field Of View (FOV) of the scene for visual tracking, and removes the need for lever-arm compensation (*i.e.* adjusting the reference frames of the tracking system to align with the user's eyes).

The improved accuracy of an AR system due to tracking also prevents problems such as visual capture [143] and does not allow visual sensors to gain priority over other sensors. For instance, inadequate registration accuracy can cause the user to reach or walk to the wrong part of the real environment because the augmentation has been displayed on another part. The eyes of the users get used to the error in

the virtual environment and after some time of usage they start to accept these errors as correct, which is not desirable.

2.7.1 Methods

A variety of vision-based tracking methods for various applications are found in literature [172]. This section provides a review of tracking methods used for AR applications under four main categories: indoor methods, outdoor methods, fusion strategies and recent approaches – see Figure 2.8.

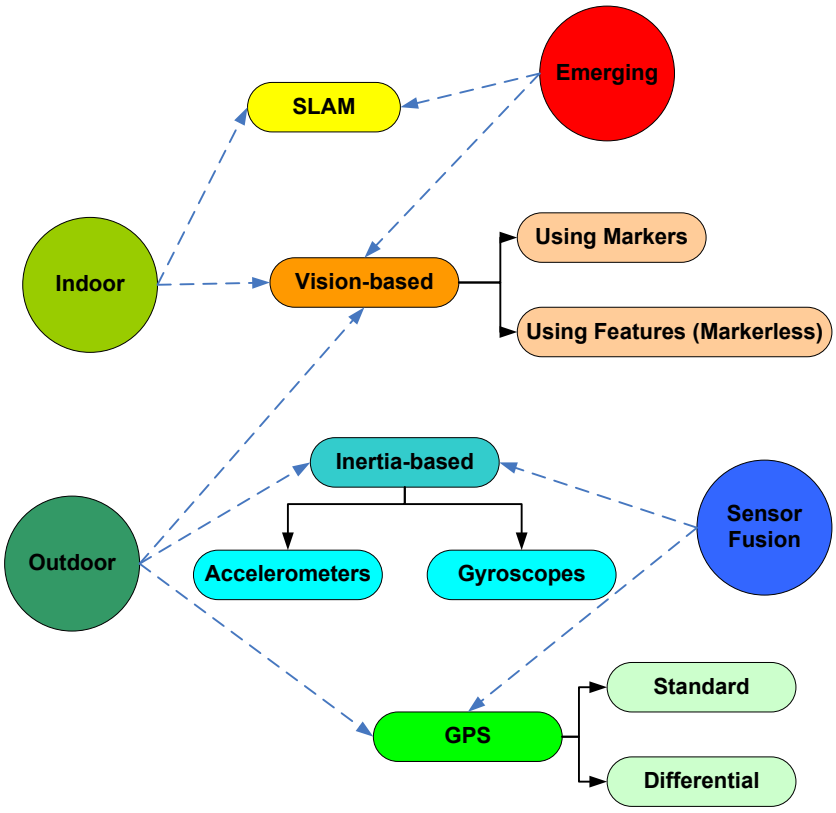


Figure 2.8: Tracking methods for AR

Indoor techniques

Indoor environments provide a structured domain for an AR application; and movements of the user are limited only to a specified region. In [151], it is stated that, for an indoor space, the dimensions of the environment are fixed and the user's possible movements are more predictable. The structured domain also provides power for the tracking equipment and presents a controlled environment [173].

Before proceeding, it is important to understand the term 'marker' used in the context of these methods. Fiducial markers are distinguishable elements put in the environment so that they can be recognized among other objects in the same environment. These markers can be categorized as *active* or *passive*. Active markers emit a signal (*e.g.* magnetic field, Radio Frequency IDentification (RFID) or light) which can be sensed by a sensor. Passive markers tend to be a pattern which can be easily isolated from the texture of the environment (*e.g.* QR codes). In this case, computer vision methods can be applied to recognize the marker. Markers are sometimes also referred as 'beacons' and 'landmarks'.

ARToolkit [174] is vision-based library for tracking square-shaped markers for camera tracking for AR. The library uses a special set of markers that can be recognized by it. When these markers are within the camera's FOV, the system can detect and identify the type of the marker and uses several of them to estimate camera pose. Once this is known, 3D models in VRML format can be overlaid on the input image. Figure 2.9 shows this process and how ARToolkit can be used for displaying ancient buildings on a table.

Indoor tracking is generally achieved by either of two methods: outside-in and inside-out as shown in Figure 2.10. The names of these methods present clues

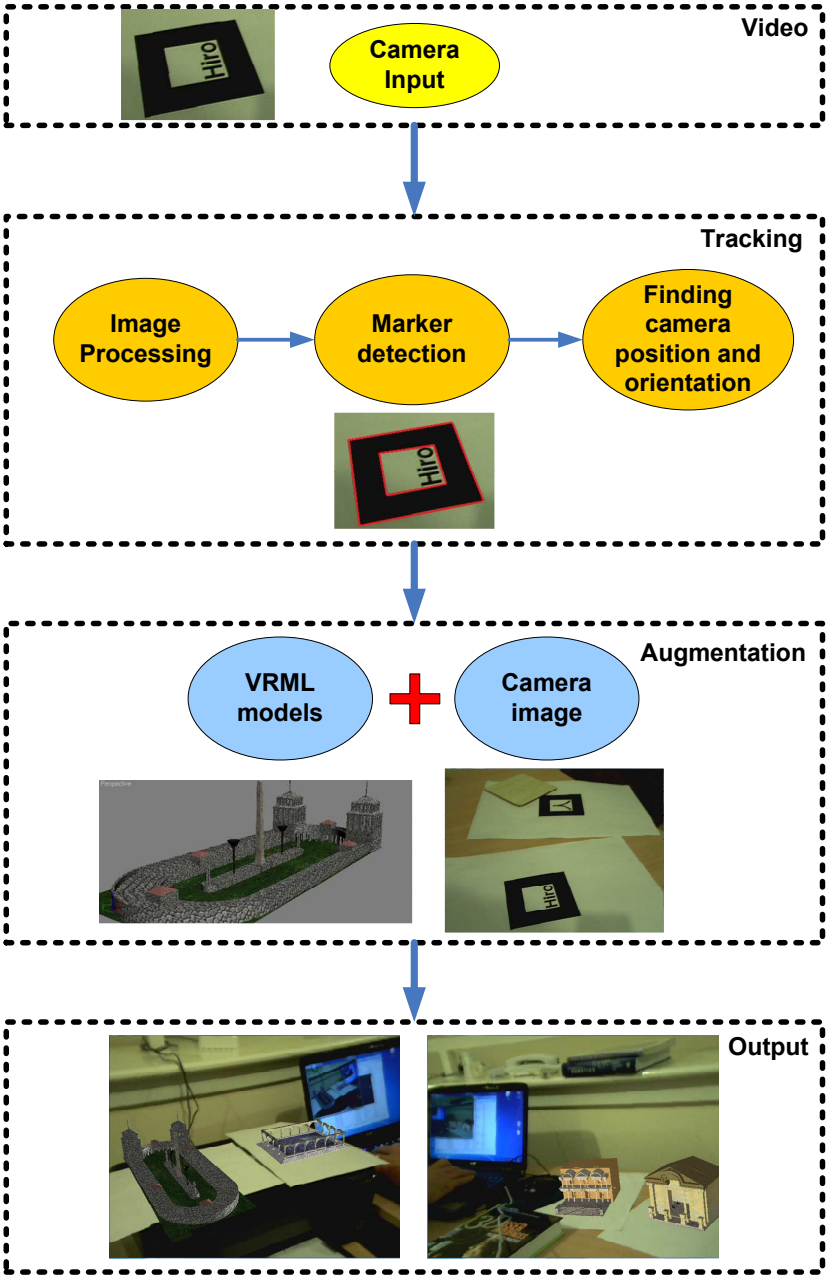
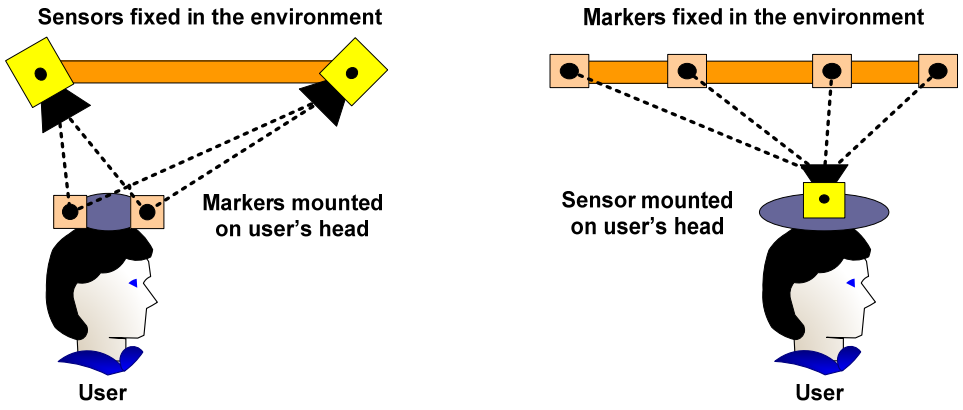


Figure 2.9: Operation of ARToolkit

about the location of the sensor, which can be magnetic, ultrasonic, RFID sensors or a camera, as well as how the tracking is achieved. In the first method, the sensor is fixed in the environment. The user wears a hat-like item on which markers are mounted. As the name suggests, the sensor is placed somewhere outside the user (outside) but is sensing the markers on the user (in). Conversely, for inside-out, the user carries the sensor and the markers are mounted around the environment (certainly within the sensor's range or FOV). As the locations of these markers are well-known in advance, tracking can be achieved.



(a) Outside-Looking in

(b) Inside-Looking out

Figure 2.10: Tracking methods based on sensor and marker positions. Following [175].

Although there are many different types of indoor tracking methods using magnetic or ultrasound sensors, these systems generally use both expensive and complex hardware [176, 177]. Although GPS is a good option for tracking user position outdoors, indoor environments such as laboratories or buildings generally attenuate or block these signals. The absence of GPS signal in indoor environments implies more reliance on vision-based tracking systems.

A wide-range tracking system named HiBall was presented in [175]. The aim of the study was an accurate, robust and flexible system to be used in large indoor environments. The HiBall device was designed as a hollow ball with a dodecahedron shape. The upper part of the device was fitted with 6 lenses to sense the IR light from Light Emitting Diodes (LEDs) mounted on the ceiling in the laboratory and controlled by an interface board.

A complete tracking system named Video Positioning System (VPS) was described in [178]. This system used fiducial markers. A new fiducial pattern design was introduced which provides unique patterns for accurate position and orientation calculation. The pattern design was based on Region Adjacency Graph (RAG) with two parts, namely *key* and *identifier*; the key indicated that this pattern is a marker, while the identifier differentiated between various markers. VPS was also applied on a parallel architecture in [179], where it was shown that parallelism improved the performance of some parts in the system for real-time operation.

In [180], a comparison of the VPS system [178] and ARToolkit was presented. The results showed that VPS provided more accuracy than the popular ARToolkit system against moderate changes in viewpoint and distance. However ARToolkit performed better when distance is increased and the authors suspected that this was due to the design of the fiducial markers.

Markerless methods can be used to extract features which are already present in the environment. These features can be points [181], lines [182] or higher-level geometric structures such as planes [10]. It is important to note that these natural features should be robust and stable so that the tracking can be performed accurately.

Chia *et al.* [181] developed a camera tracking system based on natural features.

The system used pre-captured reference images of the scene and then RANSAC was used for robust matching to achieve invariance in motions of the feature points. The system was able to run at 10Hz using some fiducial markers as well.

Park *et al.* [183] tracked several 3D objects simultaneously, robustly and accurately in real-time using a combination of object detection and frame-to-frame tracking. The latter was found to be less computationally demanding than the former but it was prone to fail and the former was more robust but it was slower. These techniques were combined to benefit from their advantages. For each target object, a 3D Computer Aided Design (CAD) model and a small set of reference images, named as keyframes, were available. Keypoint detection and pose estimation was performed in one thread and keyframe detection was performed in another thread working in parallel. The system was able to work at 15–20 frames per second (fps) on a 3.2GHz multi-core Central Processing Unit (CPU), though the performance deteriorated when the number of objects increased.

Bekel [184] presented a viewpoint-based approach for AR. This method used Self-Organizing Map (SOM) to train as a classifier which was later used to label different types of objects in form of overlays.

Adams *et al.* [185] developed a method for aligning viewfinder frames obtained from the camera of a mobile phone. They applied their system to three different cases: night-view, panorama and an input method for the camera instead of shaking. The authors stated that two algorithms were required for alignment. The first was the generation of a 'digest' by extracting edges in horizontal, vertical and two diagonal directions and a set of point features. The second part was the alignment of edges. This gave the translation between two frames. Then, by using the point feature correspondences, the confidence of the initial translation was obtained. The alignment algorithm developed in the study was fast and

robust against noise; however it was fragile against rotations, where rotations greater than 1.5° were reported to create problems.

A different approach for pose tracking with the built-in camera of a mobile phone was followed by Wagner *et al.* in [186]. They used SIFT for robust feature detection and Ferns, which is a fast classification method requiring a great amount of memory. To alleviate the computational expense of the SIFT method it was altered by removing the calculations required for scale invariance. Instead, a database of the features at different scales was used for the same purpose. FAST was used for corner detection for its high repeatability.

VisiTrack system was developed in [187] for tracking on mobile devices using point and edge extraction together with colour segmentation. Although the system was claimed to provide markerless localization, a marker can be seen in the test sequences of the system running at 25fps.

For the indoor AR system in [147], visual tracking was used. The system recognized image views of the environment acquired beforehand. Processing was carried out by remote computers via a wireless network.

Outdoor techniques

Indoor environments are more predictable whereas outdoor ones are limitless in terms of location and orientation. As opposed to indoors, there is less chance for preparing the environment to track the user while working outdoors. Moreover, predefined artificial landmarks cannot be used and natural features need to be extracted. Also, varying light poses a challenge for camera tracking which is not an issue indoors.

As mentioned earlier, GPS is considered a good tracking option when working outdoors. A comparison of GPS receivers, including different brands such as

Trimble, Garmin and DeLorme, is given in [151].

A differential GPS and a compass was used for position and orientation estimation in [164]. Latitudes and longitudes of several viewpoints were stored in a database along with the set of images taken at different times of the year with varying light conditions.

Reference images were used for video tracking and matching was performed to find these reference images for the outdoor AR system in [163]. A video image was compared with the reference images and a matching score was obtained. For the best matching score, the 2D transformation was calculated and the current camera position and orientation were deduced. This transformation was used to register the model on the video frame. The matching technique was based on Fourier transformation to be robust against changes in lighting conditions, hence it was limited to only 2D transformations such as rotation (requires additional processing) and translation. This technique had a fixed number of computations and so was suitable for real-time operation without markers, working at 10Hz.

Inertial sensing is a widely used method since its operation is similar to the otolithic stones in human ear [75]. Accelerometers are used for detecting translational motion and gyroscopes for rotational motion. This method is generally used together with other tracking methods as will be explained in more detail below.

For tracking the user's location in the ancient city of Paestum in Italy [165], a wireless system was planned, using antennas instead of GPS. However, this was not implemented due to the opposition to changes in the archaeological site by the archaeologists [163].

Tracking for a walking or running user was performed using a custom tracking system in [188]. The system used an inertial sensor, electromagnetic sensor, push-

buttons in the heels of the user's footwear and trackers at the knees, so that the motion of the legs could be obtained. The transmitter was mounted above the user's waist so that the relative motion of the legs could be extracted when the user's foot does not ground.

Fusion strategies

When the above-mentioned methods are used with a single sensor, accuracy of the tracking may be low. However, accurate systems can be obtained by using different sensors in combination, an approach known as *sensor fusion*.

Fusion methods are classified as loosely and tightly-coupled systems [75]. In loosely-coupled systems, the sensors act separately and perform calculations regardless of each other. However, in tightly-coupled systems, estimates from different sensors are directly combined to generate a single and improved position and orientation estimate.

Visual-inertial tracking is a popular technique, due to the complementary characteristics of the sensors, and is used in many different applications. Vision allows estimation of the camera position directly from the images observed [189]. However, it is not robust against 3D transformations, and the computation is expensive. For inertial trackers, noise and calibration errors can result in an accumulation of position and orientation errors. It is known that inertial sensors have long term stability problems [190]. Vision is good for small acceleration and velocity. When these sensors are used together, faster computation can be achieved with inertial sensors and the drift errors of the inertial sensor can be corrected using vision. Applications generally use low frequency vision data and high frequency inertial data [86] since visual processing is more expensive and trackers today can generate estimates at rates up to 550Hz using custom hardware [191].

A hybrid tracking system was developed by [144] for mobile outdoor AR. The system combined vision-based methods and inertial trackers. The developed inertial tracker hardware included 3 accelerometers, 3 gyroscopes and a Digital Signal Processor (DSP). These devices were used to track accelerations in x, y and z coordinates. The vision system used point features and calculated the 6 Degree of Freedom (DoF) camera pose using PnP.

Another visual-inertial tracker system was developed by Foxlin *et al.* [192]. One or more cameras could be added to the system. The authors developed a fusion filter to combine visual and inertial measurements. Circular fiducial markers were used for tracking.

A self-contained tracking system for outdoor using inertial and visual sensors was developed in [173]. The system used a fiducial design based on colours for and indoor AR application.

You *et al.* [189] developed a hybrid system for accurate registration in AR. A prediction-correction method was used in their system. The data obtained from the inertial sensor was used to estimate 2D feature motion (2D prediction) and then visual feature tracking was employed to correct the estimate (2D correction). Finally, a 3D correction was performed by the gyroscopes from the 2D motion residual.

User tracking in [151] was performed with GPS and head trackers and a camera was only used for capturing the view. System components included a Trimble AgGPS 332 Receiver, a TCM5 3-axis orientation tracker, a Wristpc wearable keyboard2, a Cirque smart cat touch pad3, i-glasses SVGA HMD, and a laptop.

Inspired by a desktop optical mouse and based on the "Anywhere Augmentation" paradigm introduced by the authors for outdoor AR, a tracking system with a camera aiming directly to the ground and an orientation tracker was developed

in [193]. The system additionally used GPS to prevent long term drift of the system.

Haala *et al.* [194] used a low-cost GPS and a digital compass for positioning in an urban environment. The authors applied shape matching with the 3D model of a building and the actual building. When the system found a match, the 3D model was overlaid in the video.

Piekarski [157] developed an outdoor AR system using a Trimble Ag132 GPS unit and an orientation tracker, and achieving an accuracy of less than 50cm. In the software, the user was able to define the corners of 3D model to be drawn with a pinch-glove. The marker on the glove was tracked by the system for this purpose.

A different outdoor application which aimed to display an archaeological site to users was given in [149]. The system used GPS together with inertial sensors provided within the HMD.

Sherstyuk [195] developed a novel method for fast semi-automatic 3D geometry acquisition using motion tracking equipment which was intended for quick surface prototyping where quality is not of high priority. A life size medical mannequin (articulated doll), having additional touch sensitivity to arbitrary locations, was used. Then a surface scanning method was used to track the motion of the user and generate the 3D reconstruction of the mannequin for medical visualization.

Emerging approaches

Finding the position and orientation of an agent is an issue for tracking in both AR and robotics. There has been a vast amount of research in the robotics field about this topic. Since SLAM algorithms can be applied for a robot, they can also be applied to a camera mounted on the user in an AR context. Conventional tracking

techniques, as explained before, have their own advantages and disadvantages in different situations. Application of SLAM algorithms to tracking brings new initiatives to current state-of-the-art systems.

An interactive and interesting application was presented in [196] where Chekhlov *et al.* applied EKF SLAM, given in [69], to an AR game in which a ninja tries to jump from one plane to another until he reaches the target plane. Higher level structures such as planes were created from point feature sets using RANSAC and the OGRE game engine [197] was used to implement the game.

Bleser [198] investigated the robustness and accuracy of real-time markerless augmented reality in both known and unknown environments for AR. Bleser used sensor fusion of IMU and a camera in a PF framework. A CAD model of the environment or an object was matched with the actual one. The tests showed that operation in a small environment of 2.1m×1.5m×2.5m, while a conceptual solution for large environments was presented.

One of the most impressive results for AR using SLAM was presented in Klein *et al.* [199]. Their system used markerless tracking in a parallel system. Two threads were executed, for tracking and mapping of the environment. They also presented interesting AR applications such as a desktop game and a magnifying glass for burning objects in the scene using heat from a virtual sun.

Kozlov *et al.* [111] proposed using AR as an approach to visualize the internal state of a robot in order to test and debug SLAM algorithms. Their approach presented a different point of view, as the methods mentioned above used SLAM for AR. The authors proposed using visualization for robot pose, state map and data association where cross correlations could be used to show the decrease of uncertainty in the map.

2.8 Problems with Current Approaches

Current tracking systems still have many problems. Two types of errors were defined in [143], namely static and dynamic errors. Before giving details about problems when using different types of sensors, these two important terms will be explained.

The errors in tracking systems are considered as static errors due to the inadequate accuracy provided by current systems. Dynamic errors are due to delays. The end-to-end system delay is the time elapsed between the time when the tracking system measures the position and orientation of the user to the time when the images appear on the display.

All of the studies given in section 2.7 area able to present good results in terms of both tracking accuracy and application domains. However, they work well only in a small environment such as a room corner [196], a desktop [199] or a whole room [198] — all indoor environments. In [200], localization was performed according to known 3D junctions and AR tests were carried out with a rectangular pattern visible in the view during processing. The system used tokens as markers in [201], although markerless tracking was claimed.

Conversely, this research will be directed towards the use of AR in outdoor environments. There are important differences between indoor and outdoor environments for vision-based tracking methods. First of all, indoor environments are generally limited in terms of location, and hence are more predictable. Fewer features are required for tracking. Second, the lighting (which is a very important factor for vision-based processing) is constant for an indoor environment, where this may vary largely due to the clouds, shades, *etc.* Lower contrast also poses a challenge for finding and tracking robust features. Another important factor

to consider is the distance for acquiring the depth information. When a feature lies closer to the camera, the camera can extract depth information after a small movement via computational stereo. However, when the feature is far from the camera, a large camera motion is required to estimate its depth. Finally, outdoor applications do not allow placing markers; natural features must be extracted for tracking.

Vision methods allow both tracking and managing residual errors. They are low cost. The problem with these methods is the lack of robustness [146]. For some applications (*e.g.* [193]), there may be a great probability of incorrect matches since the texture of the ground is mostly similar in different areas. It was stated in [146] that the structure of AR models is more difficult than VR since the former uses geometry not only for visualization but also for occlusion handling and vision-based tracking of scene features.

Also for visual tracking, the features to be used as landmarks should be invariant to changes in lighting and viewpoint. Since this is not always possible, vision-based tracking for outdoor environments is reported to be fragile [144]. Using a camera as the only sensor was found to be accurate but expensive in computation [91].

SFM methods use off-line batch processing methods [84,85] on video sequences. The 3D structure of the environment can then be extracted using non-linear minimization techniques (*e.g.* bundle adjustment). These off-line methods cannot provide the real-time performance [181] required for AR. Therefore a sequential method which performs the localization [66] will be more useful.

Standard GPS has an error of the order of 10m. However, it improves when a differential GPS or real-time kinematic correction is used. Line of sight can be a problem for the satellites in urban environments [75] or under dense tree

canopy [202]. Other problems with GPS were explained in detail in [110]. The system developed in [152] reported tracking problems occurring when GPS was used as the only sensor, and the authors suggested using sensor fusion or GPS dead-reckoning.

Double integration of data from inertial trackers cause drift errors to propagate rapidly [193]. Active tracking systems require calibrated sensors and signal sources in a prepared environment and tracking equipment can be affected by signal noise, degradation with distance and interference sources [145]. Magnetic trackers can be interfered by the ferro-magnetic objects in the environment [203]. A system with gyroscopes and accelerometers provide good bandwidth; however, it is prone to integrated errors and long term stability is not guaranteed [204].

There are several issues that must be considered for an accurate system. As with almost all other vision-based systems, incorrect feature correspondences pose a problem since low precision and recall rates were reported for feature matching with images [104]. For a vision-based approach, RANSAC is useful for the data association problem but it was reported to result in incorrect estimates occasionally [205]. SIFT or SURF descriptors are considered as an important barrier to achieve full frame-rate operation [206]. However, using these robust descriptors can help solve the problems described above.

Similar problems come into consideration in the SLAM systems used in robotics. First of all, data association, finding a unique correspondence between the feature model and the observation due to incorrect correspondences mentioned above is a problem. The second problem is linearisation due to the characteristics of current SLAM methods (*e.g.* EKF). Linearisation problems affect both the filter stability and convergence resulting in less accurate localization [207].

SLAM applications on robots usually have odometry sensors, providing speed

information which is used as a control parameter in a filtering approach. This information is not available for a moving person and the movements of the person cannot be controlled like a robot. The system in [208] generates motion commands to maximize the accuracy of a SLAM application. A better method is to use an inertial sensor to decide on the motion model in a multiple model approach like [87]. Also, modest speeds are given for a robot as 1m/s [209]. This speed is almost the same for a tourist walking in an archaeological site to examine the ruins.

A vision-based approach is promising for motion estimation. Some of the previous systems [145, 163, 164] have used methods that employ computer vision methods for tracking; however, these systems are aimed for users standing at a fixed position. For a user in motion, most methods use GPS and inertial trackers for this purpose [148, 149, 151, 152].

The idea of using different sensors together is quite useful because different sensors can counteract each other's weaknesses and provide more accuracy. Inertial sensors can be used for stability and robustness in cases of rapid motion or occlusion [91]. Even better results can be obtained with a similar approach as [210] by fusing GPS measurements with measurements obtained from vision system. Tracking accuracy and convergence speed is claimed to improve by using more sensors [211].

2.9 Remarks

This chapter presented a review of the different methods and techniques that are employed in the thesis. Starting with computer vision, through filtering methods and fuzzy logic to tracking methods used for AR, previous work was described

with emphasis on the areas that are actually used in the remaining chapters.

AR has the potential for many interesting applications, including entertainment such as the games in [142,148] or cultural heritage applications as in [149,163] when working outdoors is considered. Using vision-based approaches for locating the user in an outdoor AR environment is a new research area; it will be especially useful for this purpose if it can be achieved with the required accuracy.

According to the review presented in this chapter, the literature presents a vast amount of studies and methods targeting the problem of finding a user's position in an AR environment. Based on the problems identified in section 2.8, the following findings have been suggested [212]:

i. A fusion of different sensors within a filtering framework is a promising approach to follow.

ii. Vision-based tracking is quite useful because we already need images of the environment for augmentation. As we have this information source at hand, it is wise to use it for both tracking and overlaying.

iii. The introduction of robust detectors such as SIFT or SURF will improve the visual tracking process, yet they are considered as an important barrier to achieving full frame-rate operation in real-time.

iv. Graphics hardware or parallel implementations can be useful for performance considerations.

These suggestions will be followed in the rest of the thesis in the development of a system that will track the position of the user in an AR application similar to the one developed in [149] in a cultural heritage [162, 166] context.

CHAPTER 3

IMAGE FEATURES

Many vision applications, including visual SLAM [69, 213], 3D dense reconstruction [214] and the vision-based user tracking method described in chapter 4, rely heavily on accurate feature detection and matching. Feature detection must be robust, stable, and invariant to changes in scale and viewpoint as mentioned in section 2.1.2. Feature descriptors need to be able to characterize features uniquely. If real-time operation is desired, both detection and matching must be quick to execute [46].

There are now many feature detectors and descriptors that exhibit some of the characteristics outlined above, and this obviously leads to questions regarding which one is best. Regrettably, different detectors detect different features in the same image, so one cannot evaluate performance based on knowledge of where features should be found; instead, researchers have either had to rely on evaluating detectors and descriptors in the context of a particular application (termed

'scenario evaluation' in [215]) or employ measures that attempt to encapsulate particular properties of detectors and descriptors [216] ('technology evaluation' in [215]). Two measures have become well-established, namely *repeatability* [216], the ability of the operator to produce the same descriptor for the same feature in different images (see also [217]), and *coverage*, the distribution of detected features around the image. It is the topic of measuring coverage in images that is the focus of this chapter, an extension of work reported in [7,8]. As will become clear, good coverage is essential if the homography between images is to be determined accurately.

Considering a pair of images acquired from the same scene from different viewpoints, there will be an overlapping region in these images. This overlapping region provides feature matches. When these matches are concentrated in some parts of this region, it is generally found that the resulting homography is inaccurate, leading to *e.g.,* poor mosaicking of the images or poor performance when tracking. Conversely, when matches are widely distributed around the overlapping part of a pair of images, calculated homographies are more accurate. Hence, it is valuable to be able to measure feature coverage in images to assess whether any resulting homography calculation is likely to be sufficiently accurate.

There are already several approaches that measure and use coverage. Although some researchers have intuitively believed that well-distributed feature matches should yield to more accurate homography estimations [25, 31, 218], the first reported work that explicitly addresses it appears to be [218], in which similar numbers of Harris points were selected from each tile of a notional grid placed over the image, the aim being to obtain a uniform distribution of features. During execution, if no features were found in a region of the image, one or more tiles were selected randomly for feature extraction. A similar, though more computationally

expensive, method was proposed in [25], where the image is again divided into tiles and the Harris corners with the strongest responses were chosen in each tile.

An explicit definition of coverage was presented in [219], though the evaluation used in their study was only qualitative, based on a frame detection method in order to analyse visually the portion of the image covered by detected features.

Dickschheid et al. [220] investigated the localization accuracy of a camera system with different feature detectors, using the convex hull as an indicator of the spatial coverage of feature points. Tuytelaars et al. [221] discussed dense sampling in order to obtain better coverage of the image using simple spatial relationships between feature points for application domains similar to those mentioned earlier. Ehsan et al. [222] used the harmonic mean of feature distances as a measure of image coverage as it penalizes feature detectors that produce clustered features. There are also studies [63] that investigate pairs of feature detectors in order to obtain better coverage, using the complementarity of feature detectors [223].

The studies mentioned above involve the analysis of the spatial density of feature points in images. [219] is based on visual inspection, useless for a coverage measure. The method in [220] uses the ratio of the area of the convex hull of the feature points and the entire image, essentially the density of features. However, the approach is an average, unable to ascertain whether there are aggregations of feature points or regions with no feature points within the convex hull of the features [224, 225]. Similarly, using pairs or groups of features [63] may not be appropriate because no account is taken of the variation of feature density.

The approach used in this study is based on a robust technique used in other domains for many years [225–229] but not previously applied in computer vision. The spatial analysis approach employed here is able to ascertain at what distances the most clustering occurs, a finding that is not available in any of the studies

above. This is important because if these distances are known then it may be possible to refine the detector result simply by removing aggregated points. Furthermore, the speed of the calculation is rapid enough for on-line computation.

This work employs a statistical measure to estimate coverage and uses a null hypothesis framework to assess whether different feature detectors achieve significantly different coverages, an inherently conservative approach. The aim here is not to present this as the 'right' way to measure coverage; rather, it has been chosen as a complementary approach to those that have already been utilised. This is because, in the wider context of analysing the performance of vision algorithms, one should expect a robust algorithm, on average, to yield better performance irrespective of the datasets and performance measurement criterion used; hence, it is contended that the research community should be using a range of measures on a variety of data and determine which algorithms consistently appear good. This is analogous to measuring quantities by several different methods in the physical sciences.

The rest of this chapter starts with a brief discussion of feature detectors and descriptors providing sample feature extraction results using two images from a sequence in section 3.1. The extracted features are matched in section 3.2 in order to find feature correspondences and then obtain the correct matches using a method known as RANSAC for homography estimation.

Having extracted the features, the discussion examines spatial analysis methods, and Ripley's K-function in particular, in section 3.3, followed by a discussion of our evaluation framework for analysing the coverage of features using ANOVA and multiple range test in section 3.4. Results of the evaluation on a large set of images for several feature detectors is presented in section 3.5.

The analysis is applied to a real-world application, image stitching, in which

images of the same scene are combined into a larger image. The effect of feature coverage is assessed using the quality of the output images combined using the homography estimated from the extracted features from both images and results are presented in section 3.6. The chapter is concluded in section 3.7 with a discussion on feature detector selection for the following chapter.

3.1 Feature Detectors and Descriptors

Computer vision methods are usually based on finding prominent parts of an image known as *features*. These prominent parts are locations where pixel values are significantly different from the others in the neighbourhood. Methods known as *detectors* offer a range of means for finding these locations. A response is calculated using the method across the image and pixel positions having maximal response are selected as features.

In order to find similarities in different images, or in matching these images, a signature of the detected features must be calculated. These signatures are called *descriptors*. The descriptor of a feature in one image can be matched with features found in other images so that correspondences can be obtained across images. Based on the method descriptors are calculated, scale and rotation invariance can be achieved to some extent. Invariance to scale can be obtained by choosing features with maximal response at different scale levels whereas rotational invariance can be achieved using rotationally-invariant features such as corners, or using the number of gradient orientations in the feature neighbourhood (see section 2.1.2).

Figures 3.1 and 3.2 show the detected features using ORB and SURF detector/descriptors from two frames of a video sequence. Although the number of

selected features are the same for both detectors (the best 1000 features are chosen in each case), it is clear that the method of calculating the response function for prominent locations in the image are different. There are also similarities in the features such as the ones found on the columns.

(a) Image 1 (b) Image 10

Figure 3.1: Extraction of 1000 features from first and tenth frames of a sequence using ORB

(a) Image 1 (b) Image 10

Figure 3.2: Extraction of 1000 features from first and tenth frames of a sequence using SURF

3.2 Homography Estimation

A homography matrix models the perspective transformation between two images and represents a linear relationship between the corresponding image features, as depicted in Figure 3.3.

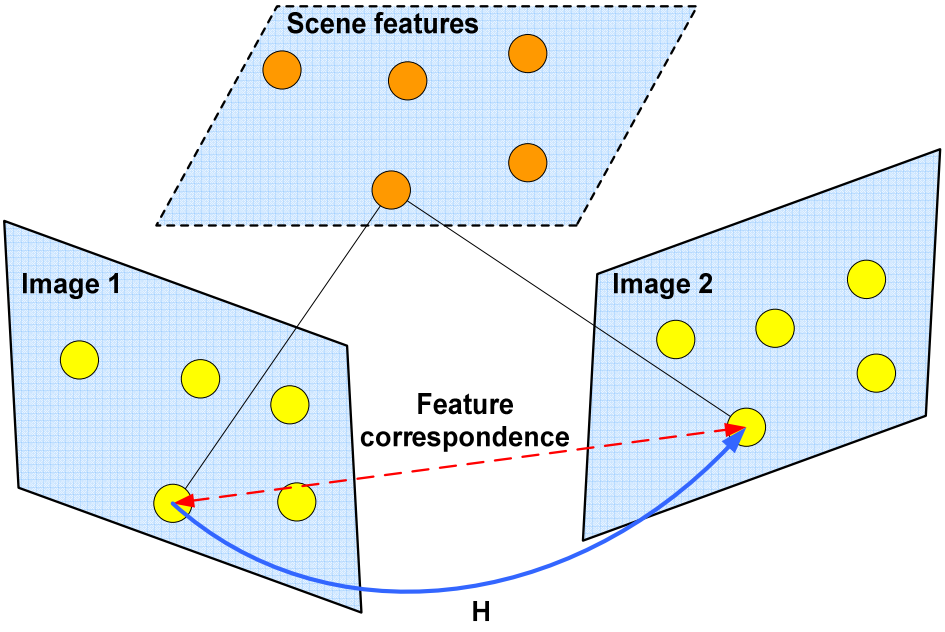


Figure 3.3: Homography H between two views of a scene

Considering a pixel position $x = (u, v, 1)^T$ in a source image and its correspondence $x' = (u', v', 1)^T$ in a destination image, both represented in homogeneous coordinates, the homography matrix will transform the first pixel position onto the second as shown in (3.1)

$$x' = Hx \tag{3.1}$$

where H is a 3×3 homography matrix

$$H = \begin{bmatrix} H_{0,0} & H_{0,1} & H_{0,2} \\ H_{1,0} & H_{1,1} & H_{1,2} \\ H_{2,0} & H_{2,1} & H_{2,2} \end{bmatrix} \tag{3.2}$$

Note that an affine transformation is obtained when $H_{2,0}$ and $H_{2,1}$ are 0 and $H_{2,2}$ is 1.

The number of elements in the matrix in (3.2) is 9 but the actual number of unknown parameters is 8 since H can be determined up to a factor of scale [31]. For this reason, only four feature correspondences are enough to estimate H — though in practice many more correspondences are used for improved robustness. Each 2D feature consists of x and y components. Thus for $N \geq 4$ feature correspondences, the homography matrix can be computed using the Direct Linear Transformation (DLT) algorithm [230] as an initial solution.

This initial solution obtained using 4 correspondences is limited, in that it may not apply to all feature correspondences. For this reason, it is normal to find many correspondences, and then use RANSAC [71] to identify the transformation that applies to them. This process can be briefly described as choosing 4 correspondences randomly from the list of all feature correspondences. Then, an algorithm (*e.g.* DLT) is used to obtain an estimate of the transformation. This transformation is applied to all of the feature correspondences in order to find the inliers (features that are compatible with the transformation). This iteration is repeated until some exit criterion is satisfied (*e.g.* maximum number of iterations) and the transformation with the highest number of occurrences is selected as the result.

This method of finding an estimate of the transformation is an important step in order to identify and discard the incorrect matches (known as outliers) from the set of feature correspondences: Outliers also have a detrimental effect on the accuracy of the position estimation algorithm described in chapter 4. After finding the homography matrix H using RANSAC, all the matching points are checked against this transformation modelled by H by applying H to the source point x. The difference d is defined as the norm of the difference between the destination point x' and the result of transformation Hx as in (3.3)

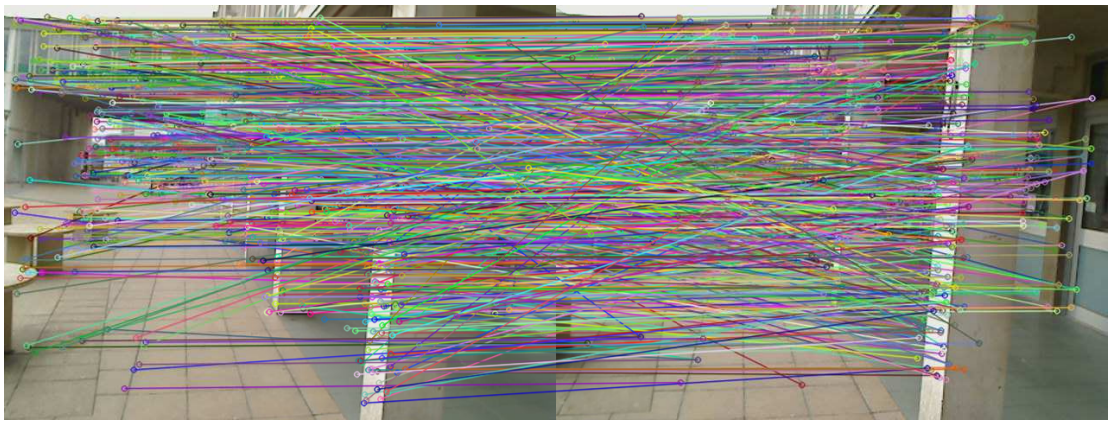
$$d = \|x' - Hx\| \tag{3.3}$$

The difference d is a measure of how far the actual and the projected points are apart from each other. In practice, values between 1.5 and 3 pixels are used since $\sqrt{5.99} = 2.45$ is given as a guideline in [31, 231] considering the probability distribution of the measurement error.

Lowe [57] suggests using the Hough transform [232] as an alternative approach to RANSAC for robust fitting since the accuracy of RANSAC degrades when the outlier rate is above 50%. On the other hand, RANSAC is faster and more space efficient than the Hough transform since the former uses a simpler model and does not require an accumulation table.

Figures 3.4 and 3.5 present a visualization of the matching process using the FLANN method [70] and removal of incorrect matches using RANSAC. The initial result of matching (Figure 3.4) show lines between pairs indicating corresponding features. It can be seen that most of the lines follow the same transformation and hence are parallel; however there are also lines connecting spurious correspondences and clearly deviating from the main trend.

(a) Initial matches using ORB

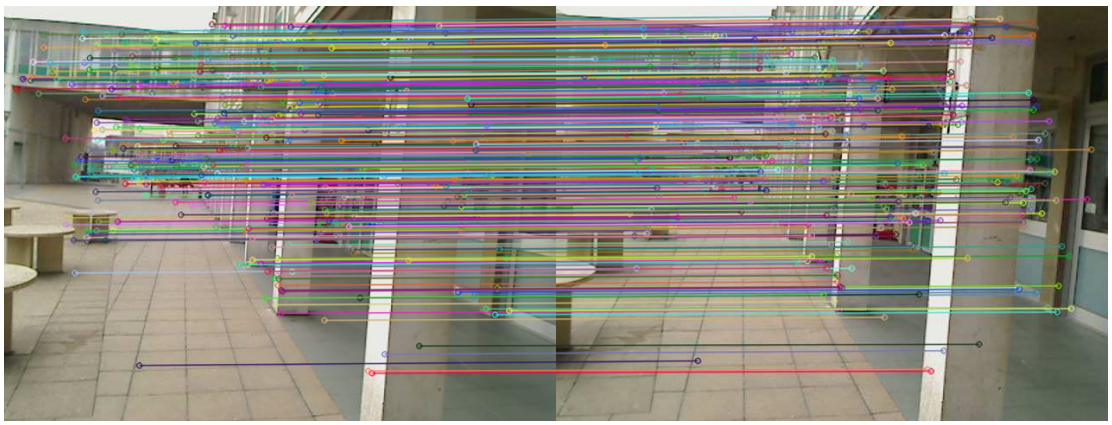


(b) Initial matches using SURF

Figure 3.4: Initial matches using ORB and SURF.

Once the correct transformation is found using RANSAC, the outliers are identified and removed from the list of matching points, leaving only lines that follow the model found as the consensus, as shown in Figure 3.5. The parallelism between the lines indicates that the remaining pairs follow the same transformation.

(a) Correct matches using ORB



(b) Correct matches using SURF

Figure 3.5: Correct matches using RANSAC to obtain the correct matches shown by parallel lines

Table 3.1 presents the number of inliers and outliers when matching was performed using ORB, which uses FAST features as described in section 2.1.2, and SURF. In the example of Figures 3.4 and 3.5, ORB has an inlier rate of 79.7%

while this rate for SURF is 42.3%. These results indicate that ORB performed better than SURF in matching features in these two images.

Table 3.1: Matching results for ORB and SURF descriptors

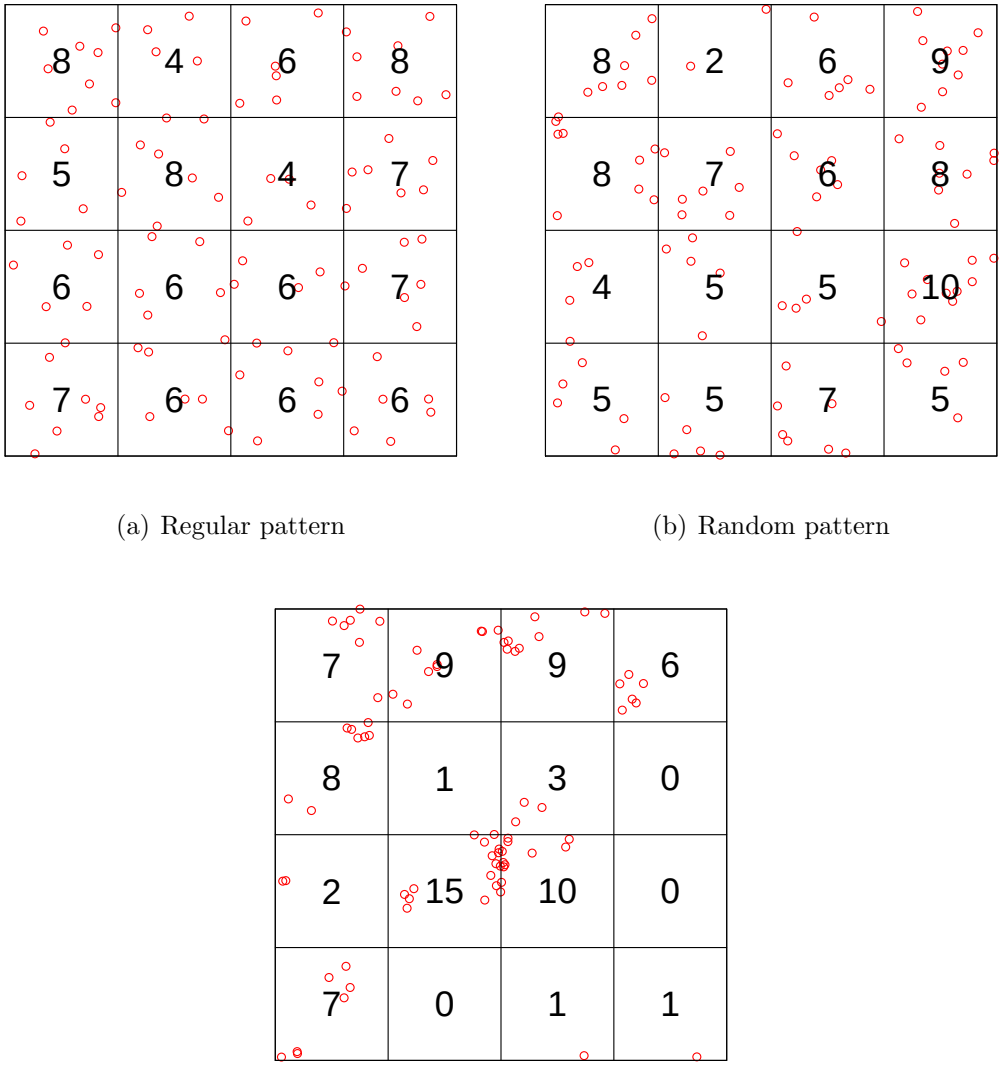
Matches	ORB	SURF
Inliers	594	423
Outliers	151	577
Total matches	745	1000

Homography estimation using RANSAC plays an important role in computer vision applications as a robust method of eliminating outliers. A direct application of homography estimation is image stitching applications where multiple images can be combined together to produce a larger image as detailed in section 3.6. Before giving further details about this application, the following section will present analysis methods used to analyse the spatial distribution of detected features.

3.3 Spatial Analysis

When one examines the locations of detected features in an image, it is typically found that there are clusters of points around particular image features; this is particularly so with multi-scale detectors such as SIFT [57] because the same feature may be found at several scales. When such inhomogeneities are present, first-order measures such as feature density are unable to describe the variation observed, yielding a poor description of the true image coverage.

Point distributions fall into three broad categories, ranging from dispersed (called "regular" in the literature), through random to aggregated (Figure 3.6). In a regular pattern, feature points are distributed uniformly over the image. Ran-



(a) Regular pattern

(b) Random pattern

(c) Aggregated pattern

Figure 3.6: Spatial patterns of feature points (following [224]). Superimposed over the patterns are 'tiles,' each of which is annotated with the number of features in that tile. The number of features in a regular pattern (a) is consistent from tile to tile; random patterns (b) have some patterns with somewhat higher or lower counts, while aggregated patterns (c) have substantially higher counts in some tiles.

dom patterns are generally assumed to follow a Poisson distribution; although visually similar to a regular pattern, clusters should be found. Finally, an aggregated pattern exhibits more clusters than would be expected to arise from a Poisson process. Note that this work does not assume or require that feature points are Poisson-distributed, though some comments on this will be made later in this chapter.

There has been considerable interest in the analysis of spatial distributions in the statistical literature, and the technique employed here is already in widespread use in ecology, such as the distribution of tree species in forests (*e.g.*, [233]), and the clustering of fast-food restaurants around schools [234] (see also [235]). In the image processing domain, it has recently been applied to the detection of microcalcifications in mammograms [236] and the stitching of mosaics [237]. A common way of analysing the distribution of spatial events is through the use of Ripley's K-function [238, 239]. In the context of this research, it may be defined as

$$K(r) = \frac{\text{number of feature matches within } r}{\lambda} \tag{3.4}$$

where r is the distance from a arbitrarily-chosen feature and λ is the density (number of features per unit area) in an image (Figure 3.7).

As $K(r)$ in (3.4) is a function of distance, it is able to describe the density of feature points at many distance scales, an important property for this work. If there are N feature points within area A and the distance between feature points i and j is r_{ij}, then one can estimate $K(r)$ as [238]

$$K(r) = \frac{A}{N^2} \sum_{j} \sum_{I, i \neq j} \frac{I_r(r_{ij})}{w_{ij}} \tag{3.5}$$

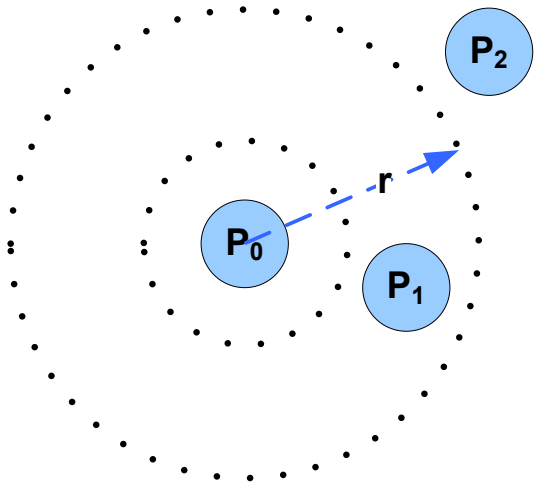


Figure 3.7: Growing the radius around a point for evaluating the K-function

where $I_r(\cdot)$ and w_{ij} determine whether a point will be included in the calculation at radius r and whether points lying on the boundaries of A are counted [224].

For a homogeneous Poisson process, one expects [233]

$$K_P(r) = \lambda \pi r^2 \tag{3.6}$$

If one plots $K_P(r)$ and an experimentally-measured $K(r)$ using (3.5), as shown in Figure 3.8, then regions where $K(r) > K_P(r)$ indicate that feature points are aggregated, while regions where $K(r) < K_P(r)$ show regular feature points. However, what is important here is not whether one is able to model the distribution of feature points as Poisson but rather whether $K(r)$ indicates that features are dispersed around the image (which is desirable for accurate determination of homographies *etc.*) or are aggregated together.

Figure 3.9 shows measured $K(r)$ and theoretical $K_P(r)$ curves for the three patterns of features of Figure 3.6; it is clear from it that $K(r)$ provides a good description of the presence of clustering of feature points in an image.

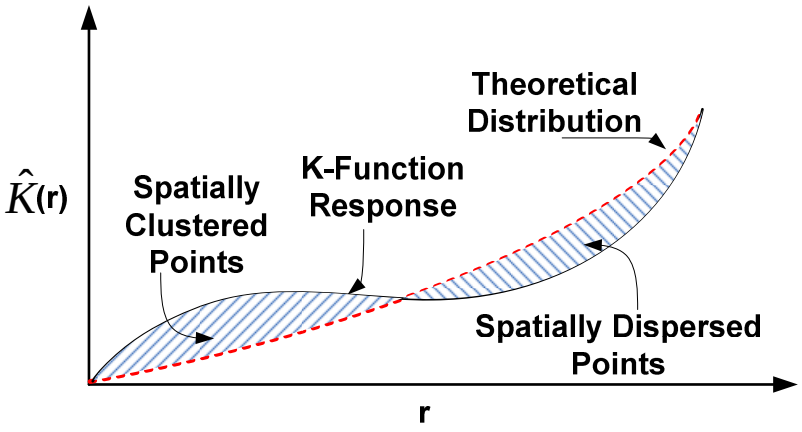


Figure 3.8: The K-functions of clustered and regular points. Clustered points produce a response above the theoretical distribution line, while regular patterns produce a response below it.

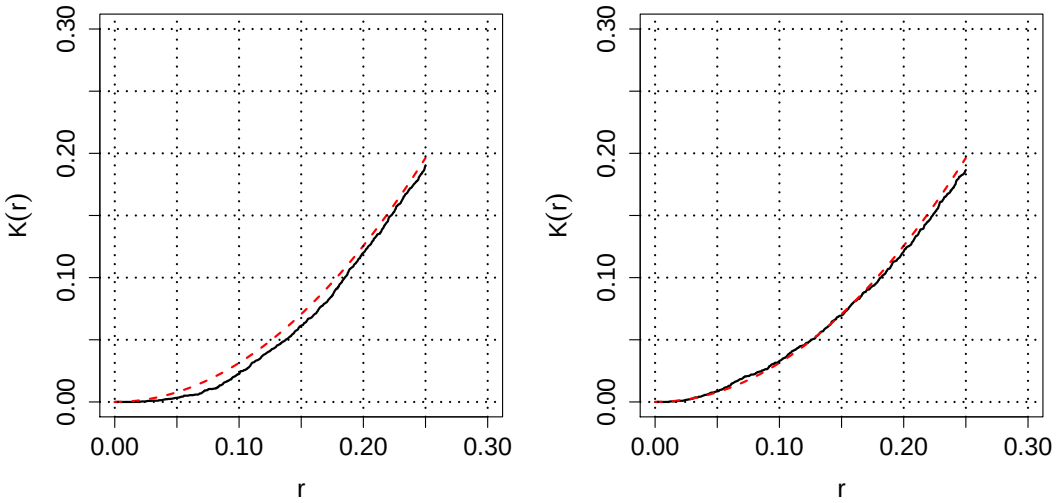
3.4 Evaluation Framework

From the discussion in the previous section, better coverage will result when the points identified by a feature detector are more widely dispersed around the image, *i.e.* $K(r) < K_P(r)$. If it should be the case that feature points are clustered at one scale but dispersed at another, as illustrated in Figure 3.8, this needs to be accommodated when assessing performance. This is most easily achieved by integrating (or summing in the discrete case) $K(r)$. For

$$\alpha = \int_{r_{\min}}^{r_{\max}} K(r)\mathrm{d}r \tag{3.7}$$

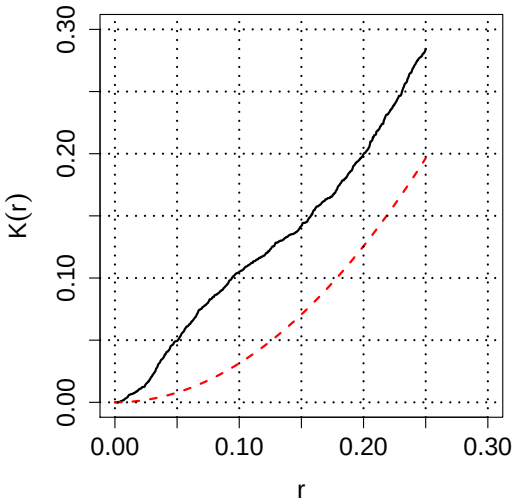
(with a similar expression for α_P, the integral of $K_P(r)$), then smaller values of

$$\alpha' = \alpha - \alpha_P \tag{3.8}$$



(a) Regular Pattern

(b) Random Pattern



(c) Aggregated Pattern

Figure 3.9: $K(r)$ for the patterns in Figure 3.6 superimposed on $K_P(r)$

indicate better overall coverage. Note that the absolute value of the difference in (3.8) was not used here as that would make it difficult to disambiguate feature detectors resulting in clustered or evenly-distributed features. This gives a straightforward approach for identifying which of a set of feature detectors yields the best coverage: for each detector, calculate α' for each image in a database of images and sum the values. The detector yielding the smallest sum of α' achieves the best coverage.

Clearly, a larger and more diverse image database should produce more reliable results. In accordance with the principles laid down in the introduction to this chapter, this work uses a newly-gathered set of 515 images for testing. An effort was made to encompass a wide range of scene types, with indoor and outdoor scenes with a variety of illumination and contrast. The images were captured using a Nikon D300 camera equipped with a Nikkor 18–200 mm lens. Images were captured in (lossless) 16-bit Nikon Electronic Format (NEF) and converted to 8-bit Portable Pixel Map (PPM) using `dcraw`[1]. The original 4288×2848-pixel images were converted to greyscale and then reduced by averaging 3×3 regions to a single pixel, resulting in 1429×949-pixel images. Some illustrative images are shown in Figure 3.10.

The feature detectors considered were an extension of those evaluated in [46]: EBR [53]; FAST [55]; Harris & Stephens [50]; HarAff, HarLap, HesAff and HesLap [46]; IBR [54]; SFOP [60]; SIFT [57]; SURF [58]; and SUSAN [49]. Publicly-available (executable) code exists for all these detectors, and all were used with their default parameter settings. Having obtained features from images using these detectors, all subsequent processing was performed using R [224] and its `spat-stat` library.

[1] Available at `http://www.cybercom.net/~dcoffin/dcraw/`

Figure 3.10: Sample images from evaluation dataset

3.4.1 Identifying significant performance differences

Although one could plot $K(r)$ for all the detectors under consideration and simply look for differences, this does not assess whether performance differences are likely to be statistically significant. As all the feature detectors were evaluated on the same image database, a more attractive approach is to use ANOVA, a generalization of Student's t-test. There are several assumptions that have to be met when using ANOVA, the most important of which are that the variances resulting from the different 'treatments' are the same, that the treatments involved are independent, and that Normal statistics apply. The latter is likely to be the case due to the Central Limit Theorem but the former two need to be examined experimentally. The first-mentioned will be addressed in the following paragraphs, and the independence of treatments in Section 3.5.

To establish this required homogeneity of variances, a generic logarithmic transformation [136] of the form

$$y = \ln(x) \tag{3.9}$$

was performed on the data (the detector results for the evaluation criterion defined above) so that the data points are squeezed together and show less variation. This transformation does not guarantee that the resulting points will satisfy the homogeneity assumption, so Hartley's $F_{\max}$ test [137] was employed on the detector results. This test involves calculating the ratio of the largest variance to the smallest (to yield $F_{\max}$) and then consulting a table of critical values [138] for the given number of treatments and degrees of freedom. The calculated $F_{\max}$ value was smaller than the value in the table ($F_{\max} = 1.021 < 1.790$), so the conclusion is that the variances can be assumed be consistent with the constraints of

ANOVA.

ANOVA is normally employed within a null hypothesis framework: one assumes the various detectors yield similar spatial distributions and assesses the evidence to accept or reject that assumption. In this evaluation, the null and alternative hypotheses are defined as

$$\begin{aligned} H_0 &\equiv \alpha'_1 = \alpha'_2 = \cdots = \alpha'_{12} \\ H_1 &\equiv \exists i : \alpha'_i \neq \alpha'_j, (i,j) \in (1, \cdots 12), i \neq j \end{aligned} \tag{3.10}$$

The null hypothesis H_0 suggests that different feature detectors result in an equal coverage (α' values for the 12 feature detectors are either equal or very close to each other), whereas the alternative hypothesis is that at least one of the feature detectors performs differently. In this evaluation there are two independent variables, images and feature detectors, while the dependent variable is α' as defined in (3.8). Hence, two-way ANOVA without replication was used for the evaluation with $p = 0.05$, so that there is a one-in-twenty chance that the findings could arise from features of the data.

3.4.2 Multiple range test

As discussed earlier, the criterion for choosing which detector performs best is to determine α' for each image in a database, sum the values together, and choose the minimum. In doing so however, one must ensure that any difference obtained is statistically significant. For this purpose, Duncan's *multiple range test* [139] was employed. This procedure is based on the comparison of the range of a subset of the sample means with a calculated least significant range.

To perform this test, s_d, the standard deviation of the difference between any

two means, is first calculated. This is used to calculate

$$Q_{s_d} = Q_{\text{val}} s_d \tag{3.11}$$

with Q_{val} obtained from tables [138] for a given number of degrees of freedom and treatments. The value of Q_{s_d} is used to decide whether or not a difference is statistically significant.

Next, the means are sorted into descending order of magnitude and the top and the bottom means of the sorted list are taken. If the difference between them is greater than Q_{s_d}, then one concludes that the difference is significant; otherwise, it is not. This is repeated by comparing the second-largest value in the table with the lowest one; and so on, until all treatments have been compared with the lowest one. As the aim is to compare every combination of detector pairs, the smallest mean is then removed from the list and the above procedure is repeated with the remaining list elements.

3.5 Evaluation Results

The detected features and resulting variation in $K(r)$ from one typical image of the database for each of the feature detectors considered are shown in Figures 3.11 to 3.22.

The numbers superimposed on regions of these figures give the number of features found within each region. This is intended as a guide only; a single value of $K(r)$ was calculated for the entire image. In each graph, the dashed red line shows $K_P(r)$ and the full line $K(r)$. From this single example, it is clear that the performance of the detectors can differ substantially. However, the aim here

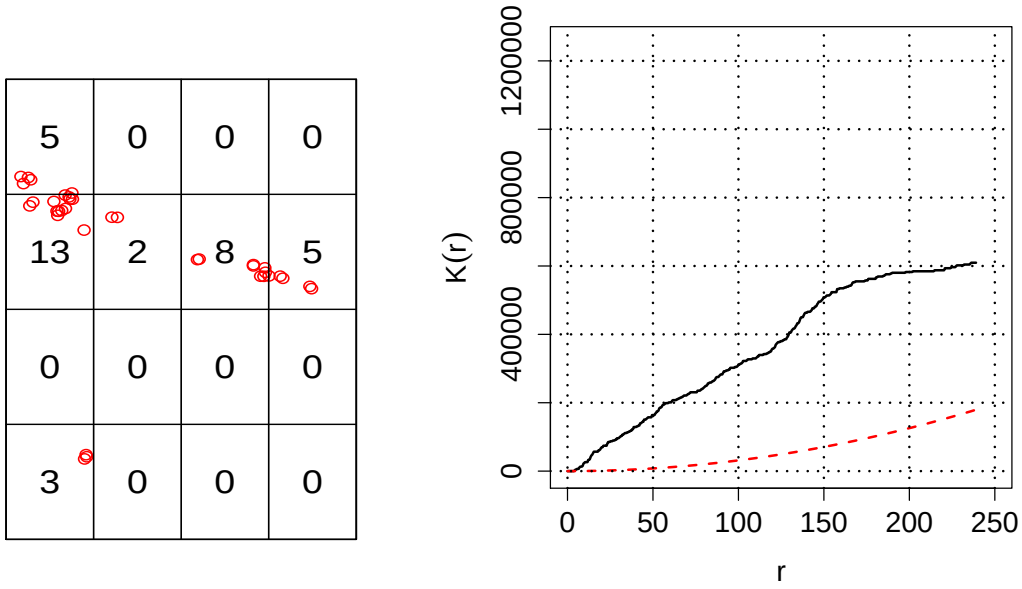


Figure 3.11: Grid counts and K-functions for EBR

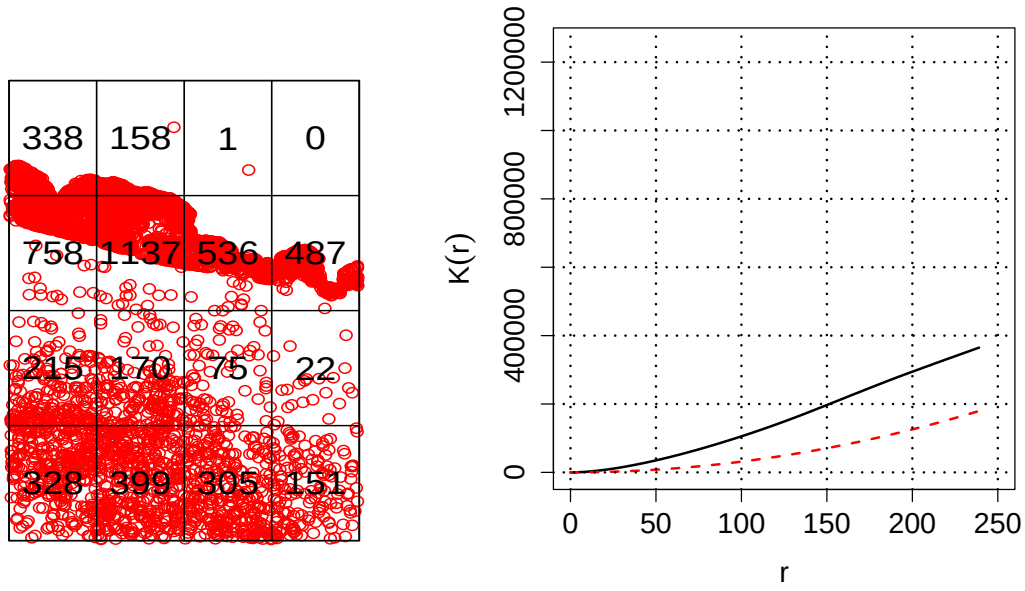


Figure 3.12: Grid counts and K-functions for FAST

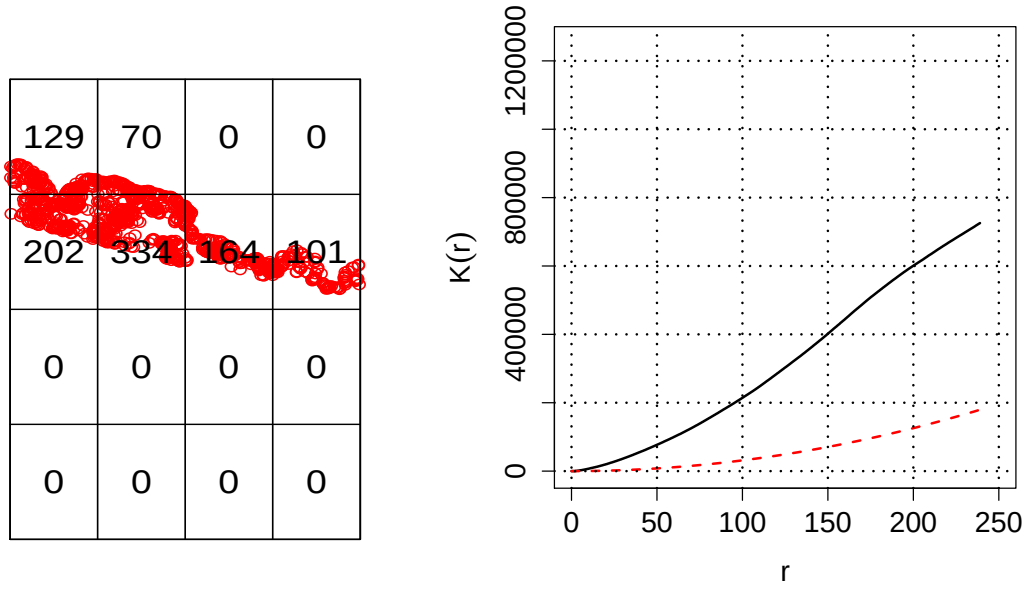


Figure 3.13: Grid counts and K-functions for Harris & Stephens

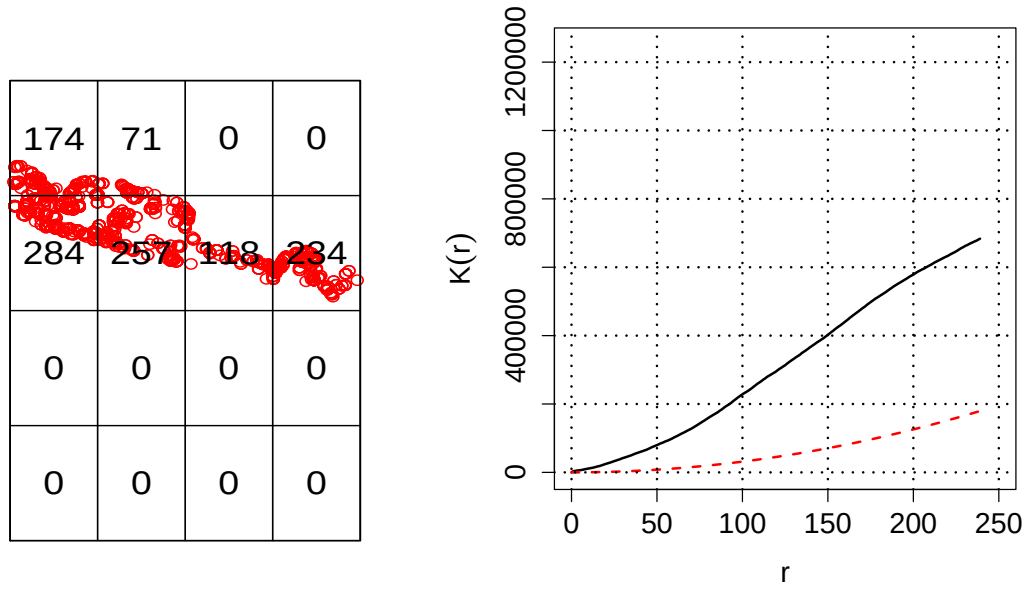


Figure 3.14: Grid counts and K-functions for HarAff

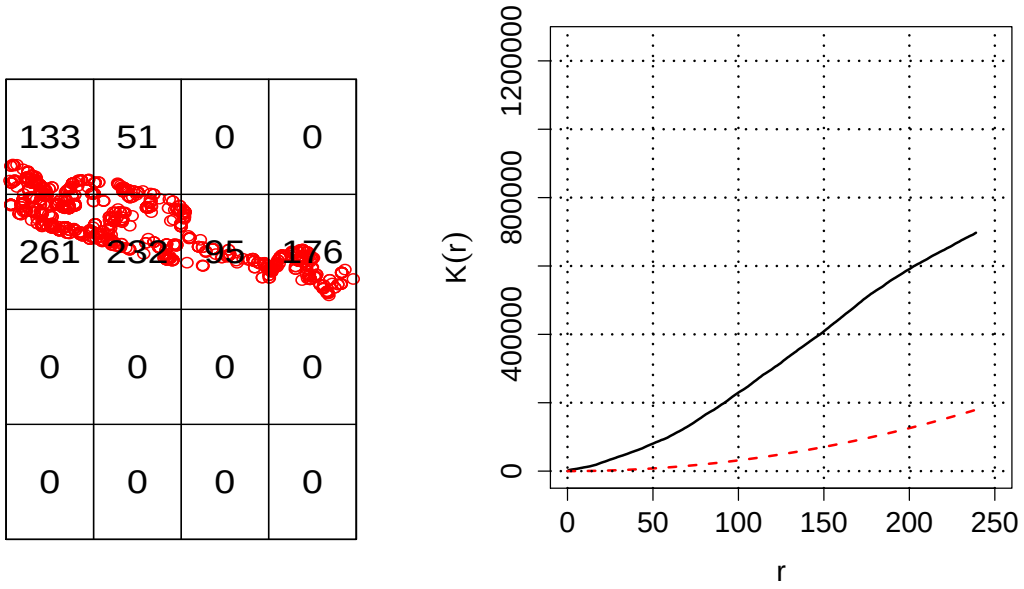


Figure 3.15: Grid counts and K-functions for HarLap

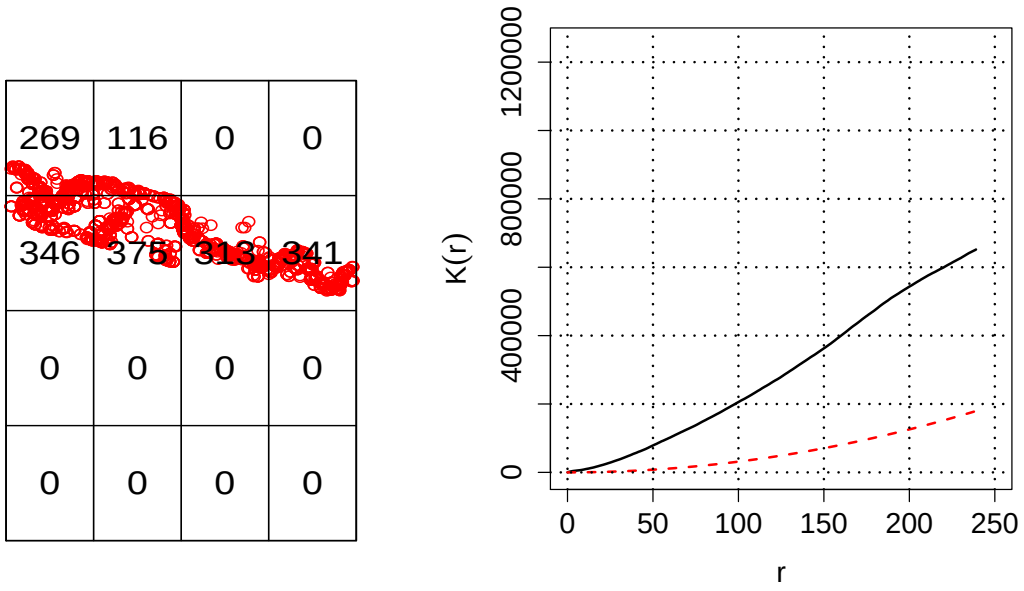


Figure 3.16: Grid counts and K-functions for HesAff

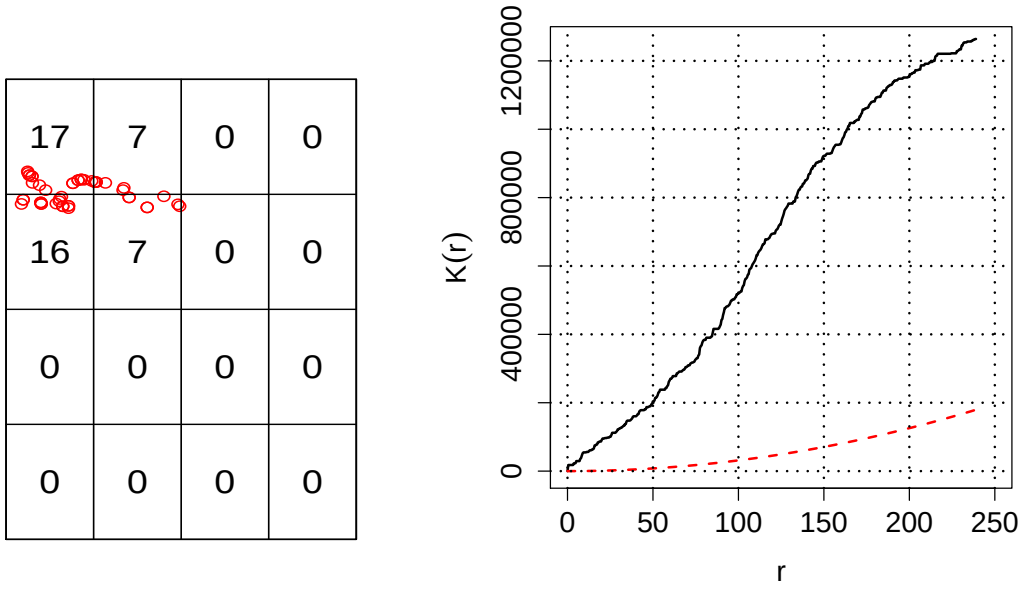


Figure 3.17: Grid counts and K-functions for HesLap

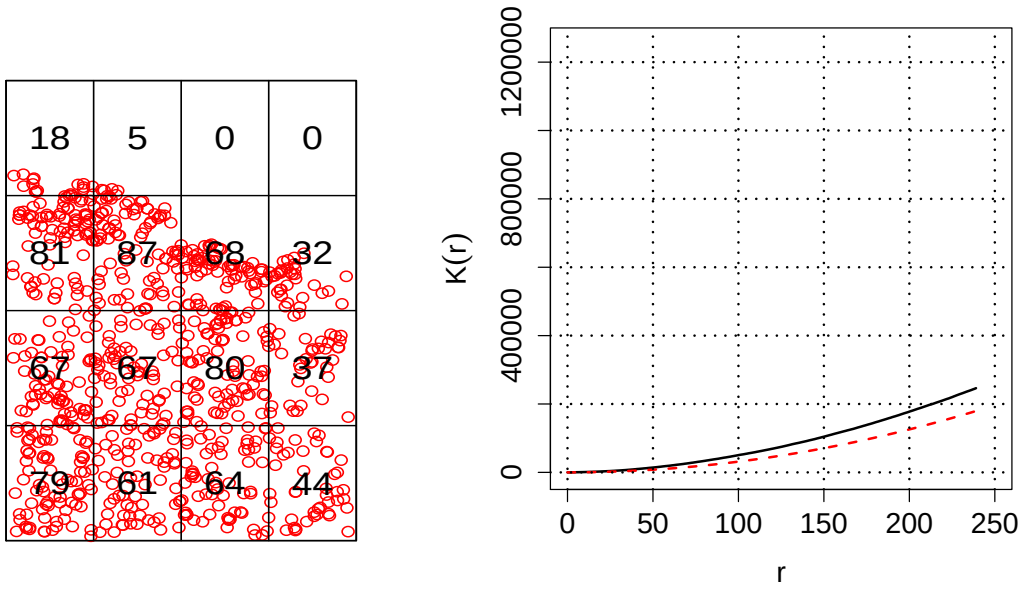


Figure 3.18: Grid counts and K-functions for IBR

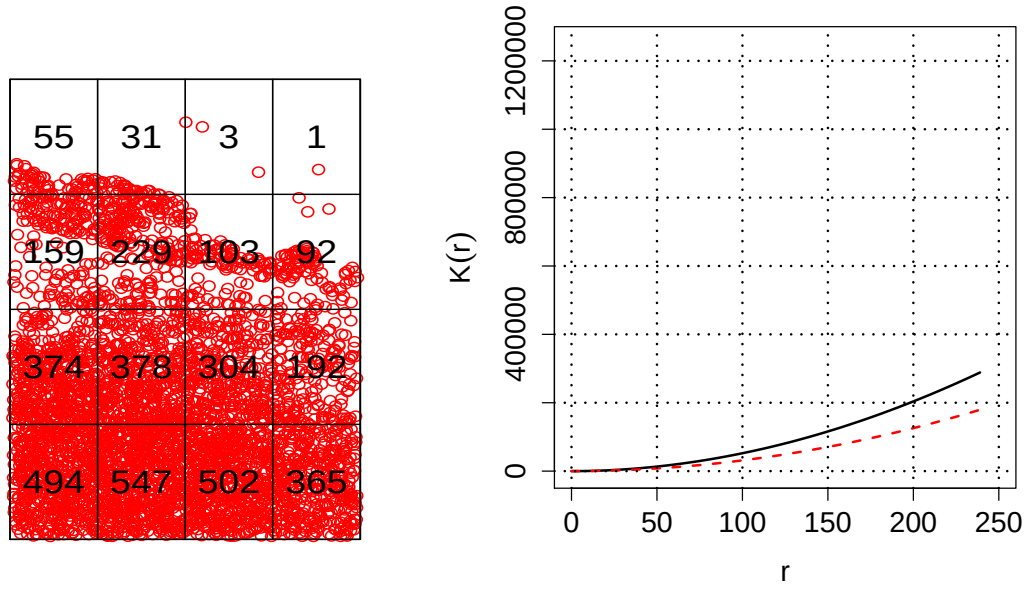


Figure 3.19: Grid counts and K-functions for SFOP

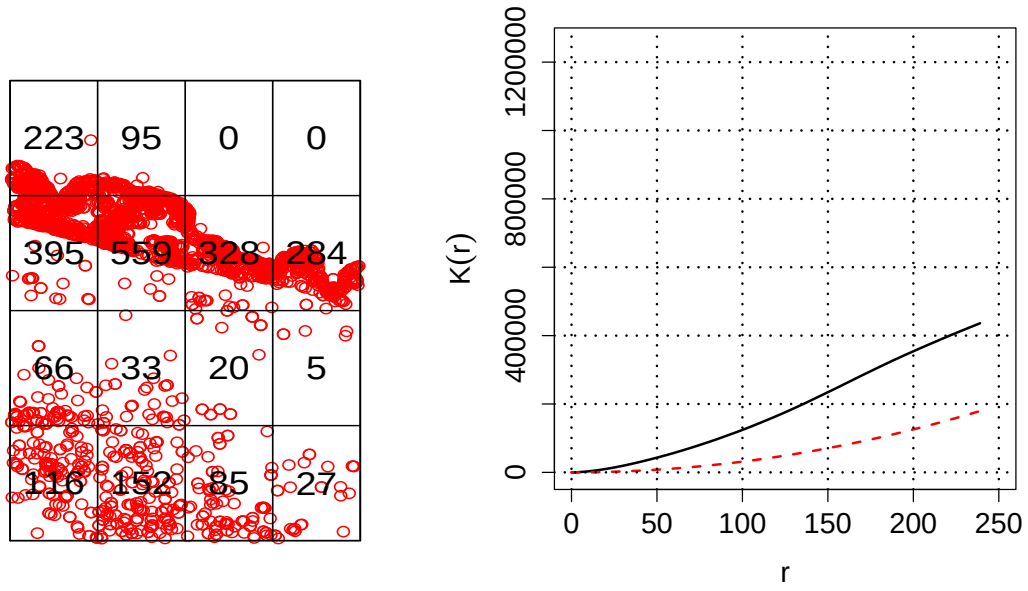


Figure 3.20: Grid counts and K-functions for SIFT

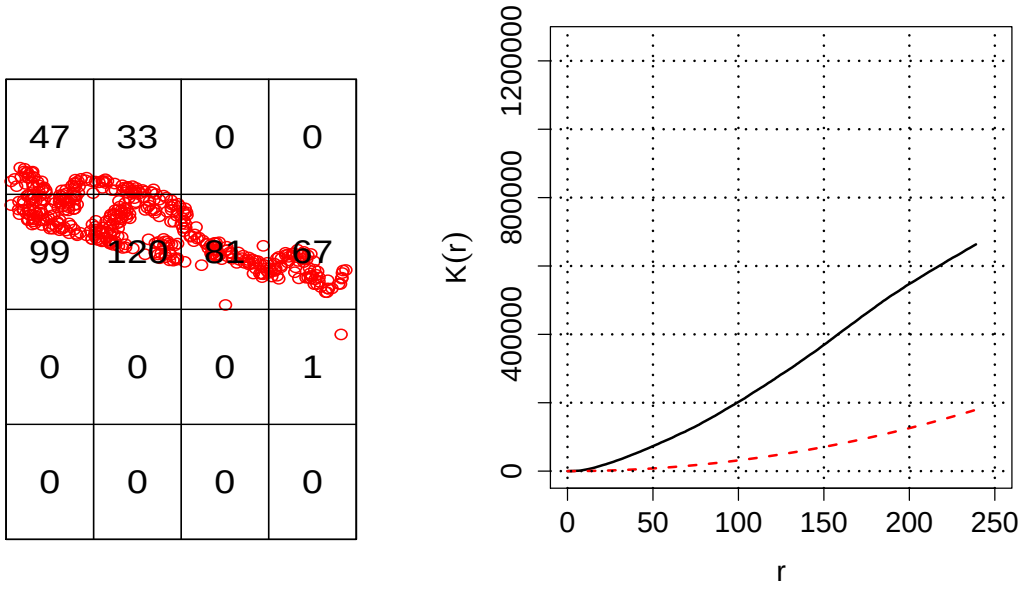


Figure 3.21: Grid counts and K-functions for SURF

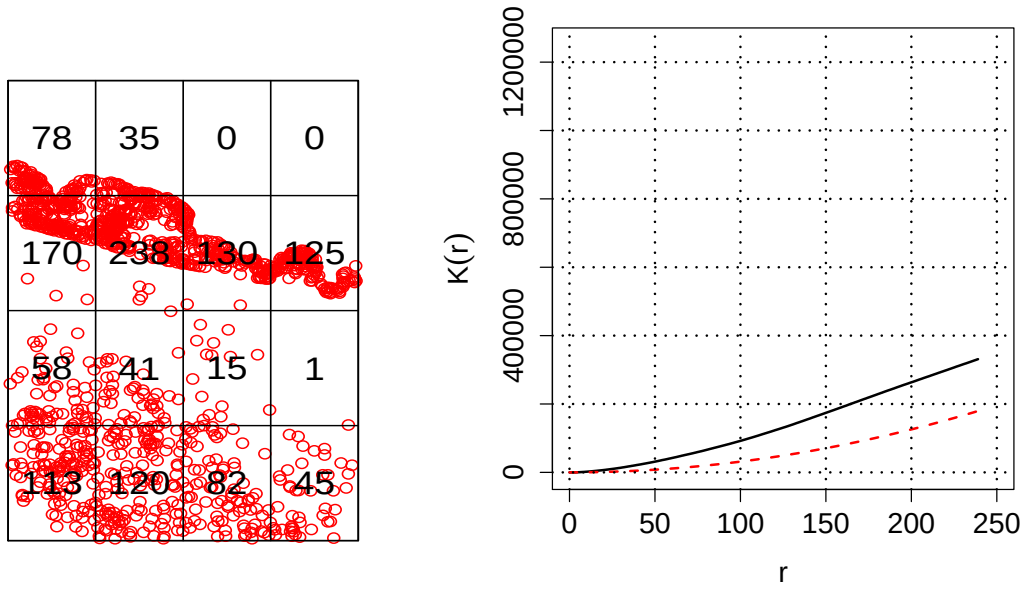


Figure 3.22: Grid counts and K-functions for SUSAN

is not to decide whether there are performance differences by inspection but to determine whether any differences are statistically significant, as determined using ANOVA.

The results of the entire evaluation exercise are summarized in Table 3.2, where the homogeneity of variance confirmed in the earlier $F_{\max}$ test is clearly visible. The proximity of the mean and median values in the table indicates that the data are evenly distributed around the mean.

Table 3.2: Feature detector evaluation results (in order of decreasing variance)

Detector	Mean	Median	Variance
IBR	15.7574	15.7932	1.5100
SURF	16.4002	16.3894	1.5089
HesAff	16.8054	16.8530	1.5080
HarAff	16.6006	16.7535	1.5079
EBR	16.9497	16.8731	1.5068
SIFT	15.9797	16.0828	1.5060
Harris	17.5607	17.6470	1.5055
SFOP	15.3130	15.3786	1.4952
HesLap	17.8867	17.9302	1.4922
SUSAN	16.6132	16.8873	1.4813
HarLap	16.5286	16.6276	1.4796
FAST	16.0879	16.2916	1.4783

Initial testing with one-way ANOVA showed significant differences between detectors but with rather large residuals. Subsequent examination using two-way ANOVA (Table 3.3) with both images and detectors as sources of variation resulted in more acceptable residuals. Both of these variations are larger than the critical value F_{crit} and hence are significant. More importantly, the differences between detectors ($714.60 \gg 1.79$) are much more significant than the differences between images ($22.10 > 1.11$), allowing one to conclude that the variation is mostly due to differences in the performance of the detectors rather than the

inherent variations in the images — this is the 'independence of treatments' assumption inherent in ANOVA that was mentioned in Section 3.4.1. Hence, the null hypothesis stating that the feature detectors are performing equally can be safely rejected. The P column shows that one can be confident that there is a relationship between the independent variables and the dependent variable about these results as $P < 0.05$. Furthermore, one can also see that values for the MS (mean square) column are significantly different ($269.61 \gg 8.34$), indicating that the effects of the independent variables (*i.e.,* images and detectors) do not interact with each other.

Table 3.3: Results of the 2-way ANOVA test

Source of variation	dof	MS	F	P	F_{crit}
Images	514	8.34	22.10	0	1.11
Detectors	11	269.61	714.60	0	1.79
Error	5654	0.38			

Table 3.4: Differences in mean performance of operators. Values marked with a star indicate that the difference is not statistically significant.

	SFOP	IBR	SIFT	FAST	SURF	SUSAN	HarLap	HarAff	HesAff	EBR	Harris	HesLap	$Q(s_d)$
HesLap													
Harris												0.33	0.11
EBR											0.94	0.61	0.13
HesAff										0.14	0.76	1.08	0.14
HarAff									0.19	0.34	0.95	1.27	0.15
HarLap								★ 0.01	0.21	0.35	0.96	1.29	0.15
SUSAN							★ 0.07	★ 0.09	0.28	0.42	1.03	1.36	0.16
SURF						0.13	0.20	0.21	0.41	0.55	1.16	1.49	0.16
FAST					0.31	0.44	0.51	0.53	0.72	0.86	1.47	1.80	0.17
SIFT				0.11	0.42	0.55	0.62	0.63	0.83	0.97	1.58	1.91	0.17
IBR			0.22	0.33	0.64	0.77	0.84	0.86	1.05	1.19	1.80	2.13	0.17
SFOP		0.44	0.67	0.78	1.09	1.22	1.29	1.30	1.49	1.64	2.25	2.57	0.18

Duncan's multiple range test was applied to decide whether the differences are significant or not. The results are summarized in Table 3.4, which shows the statistical significance of the results. Values in the final column of the table were calculated using (3.11). Each difference was compared with the value in the Q_{s_d} column; if larger, it means that the performance of the operator listed to the left is statistically better.

From the table, it can be seen that SFOP produces less aggregated patterns and gives closer results to a regular distribution of features across images over the database used, and does this significantly better than all other detectors. SFOP is followed by IBR, SIFT, FAST and SURF in increasing order of producing aggregated feature points. The results also reveal that the differences between SUSAN, HarLap and HarAff are not statistically significant. The rest of the feature detectors (HesAff, EBR, Harris and HesLap) were found to have significant differences and produce less regular (more aggregated) feature points.

Table 3.5 orders the feature detectors by their mean α', with a smaller α' implying better performance. It is instructive to compare this rank ordering with that in [220]: there, SFOP provides the best coverage, while SIFT is better than most of the other detectors considered in [220]. HarLap, HarAff, HesAff and HesLap were ranked similarly to [220]. As this work uses a different way of measuring coverage, employs different imagery and uses a statistical test to identify performance differences, yet still achieves a broadly similar ranking of detectors to [220], one can be reasonably confident that this rank ordering genuinely reflects the detectors' performances.

Table 3.5: Feature detector evaluation results (sorted by mean according to the evaluation criterion and for the multiple range test)

Feature Detector	Mean α'
SFOP	15.313
IBR	15.757
SIFT	15.980
FAST	16.088
SURF	16.400
HarLap	16.529
HarAff	16.601
SUSAN	16.613
HesAff	16.805
EBR	16.950
Harris	17.561
HesLap	17.887

3.6 Image Stitching Performance

To assess the practical value of having a good coverage of detected features and confirm the findings of the previous section, experiments were conducted on stitching images into mosaics. Here, one combines images with overlapping regions to produce higher-resolution images such as panoramas [240], the aim being to obtain a seamless transition from one image to another, avoiding ghosting or blurring effects in the panorama.

In conventional image stitching applications, one takes photographs of a scene from different viewpoints and then combines these photographs as shown in Figure 3.23.

The approach uses the homography matrices from the centre image to the left and right images. The homography matrices are computed using the feature correspondences from both images. Note that the minimum bounding rectangle of

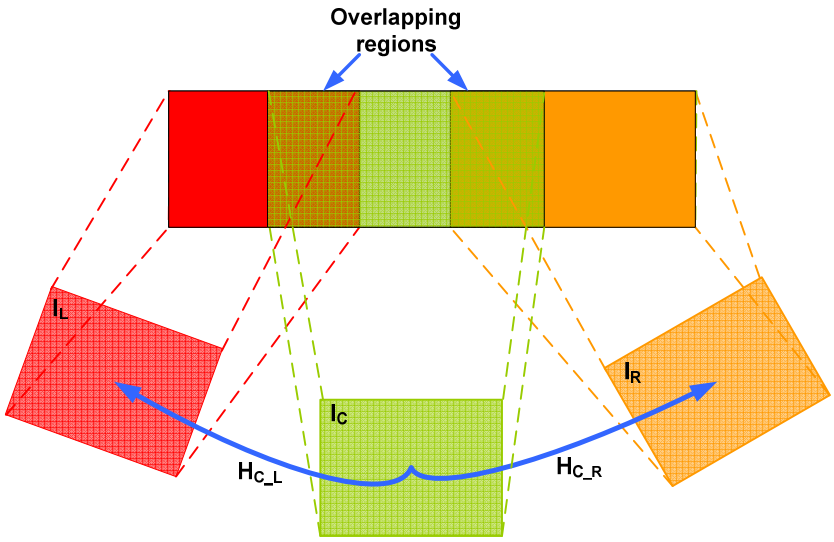


Figure 3.23: Using homography for image stitching. The homography matrices from the centre image I_C to the left and right images (I_L and I_R respectively) are computed from feature correspondences defining the overlapping region.

these correspondences will define the overlapping region (assuming there is a good coverage of features in this region). Then, the inverse of these homographies are applied to the left and right images so that their overlapping regions are aligned with the image in the centre:

$$I_F = H_{C\_L}^{-1} I_L \oplus I_C \oplus H_{C\_R}^{-1} I_R \tag{3.12}$$

where the $\oplus$ operator takes the maximum of the two pixel values to produce the final image I_F which, obviously, has a higher resolution than all the individual images.

Using this idea, an initial experiment was performed to analyse the stitching result visually using a simple image divided into three smaller images as shown in Figures 3.24(a–c). Correspondences of features detected by HesLap and SFOP were found and the homographies calculated using RANSAC. With these homo-

graphies, the stitching was performed to obtain Figures 3.24(d–e).

(a) Left image (b) Centre image (c) Right image

(d) Stitching result using HesLap (e) Stitching result using SFOP

Figure 3.24: Example of a conventional image stitching process for testing the effect of coverage. Images in (a–c) are stitched together. Note the misalignment in (d) around the gate when the homography was computed using HesLap which is not visible in (e) where SFOP was used.

The aim of the evaluation in this chapter was to propose a quantitative metric that does not rely on visual assessment and can be calculated automatically. It can be seen that the initial approach described above requires some manual work (*i.e.* dividing the image, *etc.*) in order to obtain the ground truth infor-

mation. Furthermore, this simple division only included a translation, which is not adequate for a reliable assessment. For this reason, a different, yet similar in principle, method was used so that it would be possible to compare the output image with ground truth information.

The experiment used the publicly-available *Oxford Buildings* dataset[2]. Several images of size 1024×768 pixels were selected and regions of 512×384 pixels extracted using bilinear interpolation with rotations of 10, 20, 30, 45, 60 and 90 degrees, as shown in Figure 3.25.

Features from the detectors evaluated in the previous section were matched on both the original images and the extracted regions and these were used to calculate the homography required to re-align them. The realignment is performed by warping the extracted region using the inverse of the homography, which models a translation and a rotation. Following realignment, the output image was produced by selecting the larger of the pixel values in the overlap region.

Figures 3.26 and 3.27 show the stitching results in cases where the coverage was poor and good respectively. The clusters of features and areas devoid of them are apparent visually in Figure 3.26(a), and this results in the $K(r)$ of Figure 3.26(b); this can be contrasted with the better distribution of features and the corresponding $K(r)$ in Figure 3.27(a) and (b).

The effect of poor coverage on the stitching result manifests itself principally as ghosting in the stitched image of Figure 3.26(c), while the image in Figure 3.27(c) has no such artefacts. To confirm the visual findings, Figures 3.26(d) and 3.27(d) show normalized cross-correlations between the original and stitched images, and the values are higher when the coverage is better.

[2]Available at `http://www.featurespace.org/data`

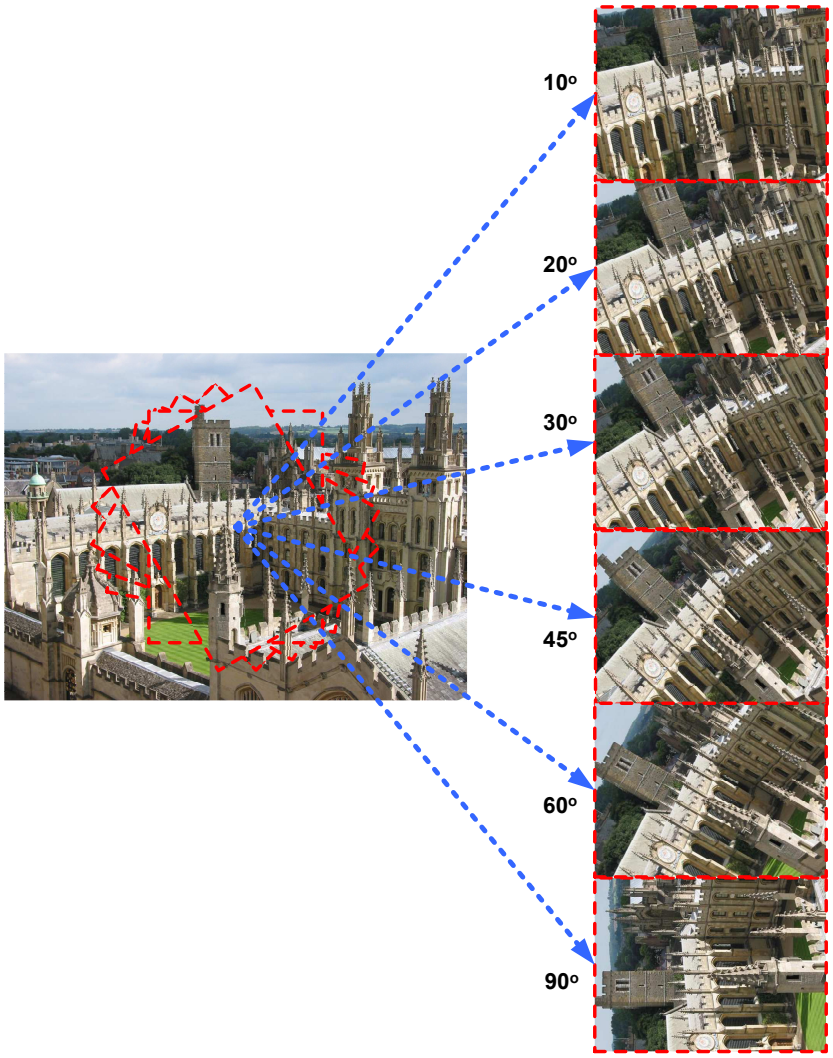
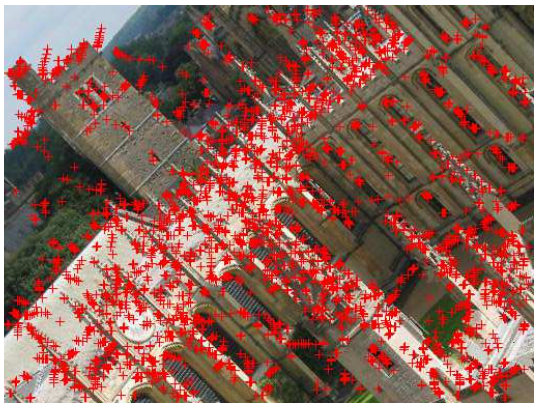
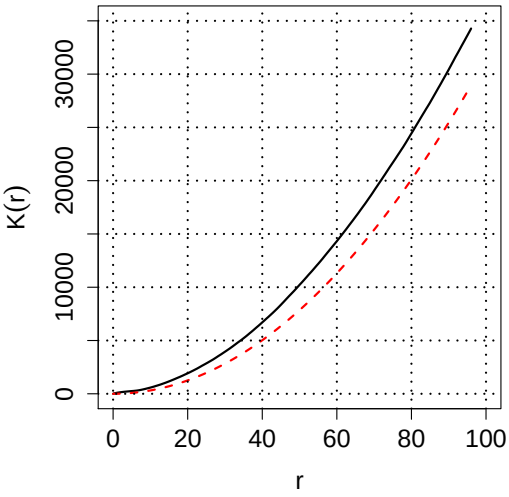


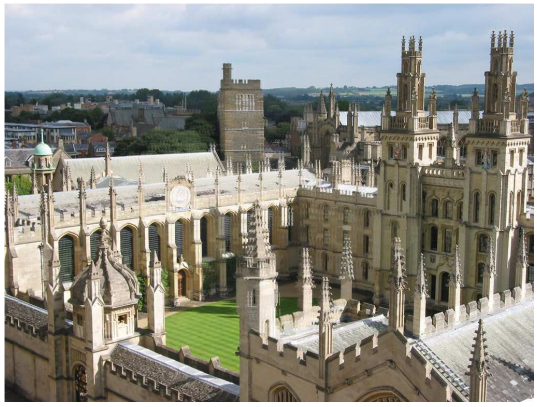
Figure 3.25: Extraction of regions with different rotations



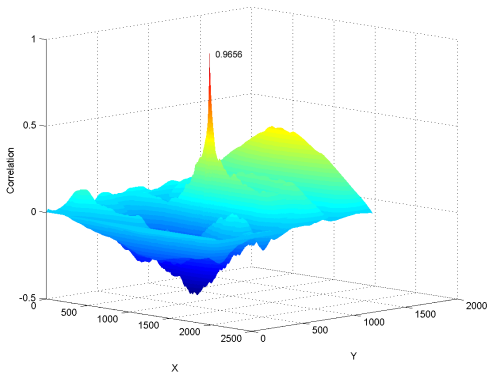
(a) Detected features



(b) Result of K-function

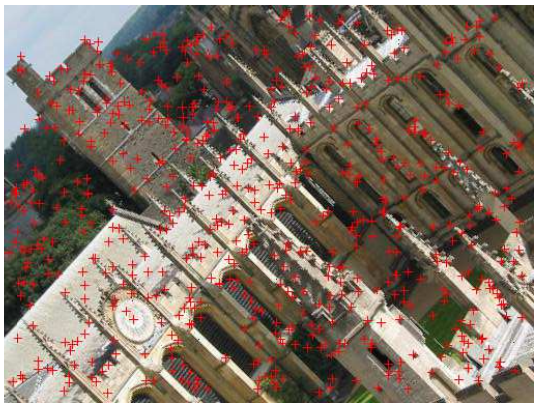


(c) Stitching result

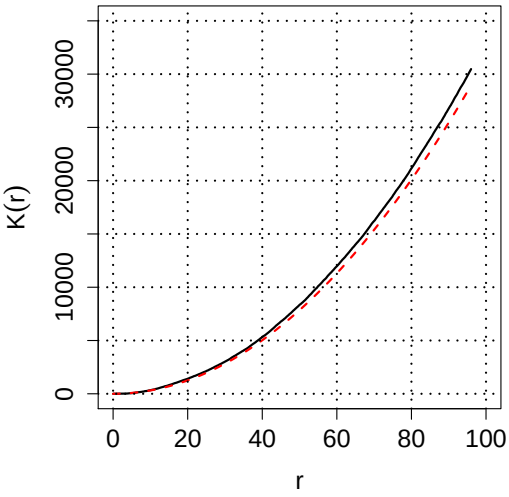


(d) Correlation plot

Figure 3.26: Stitching results when the coverage is poor

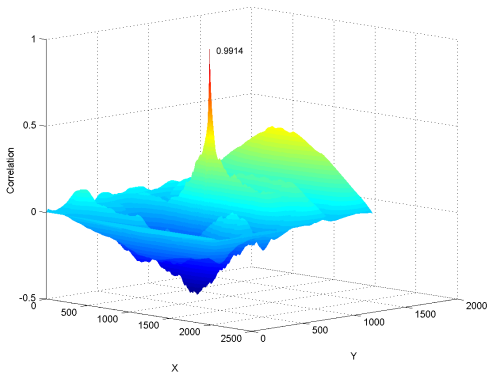


(a) Detected features



(b) Result of K-function

(c) Stitching result



(d) Correlation plot

Figure 3.27: Stitching results when the coverage is good

Figure 3.28 presents the estimated coverage values against these cross-correlation values across the entire database. It can be seen that as the coverage increases (worsens), there is a corresponding decrease in the cross-correlation. The best-fit line through the data is shown on this figure; while most of the points do not lie on this line, it is clear that the general trend is consistent.

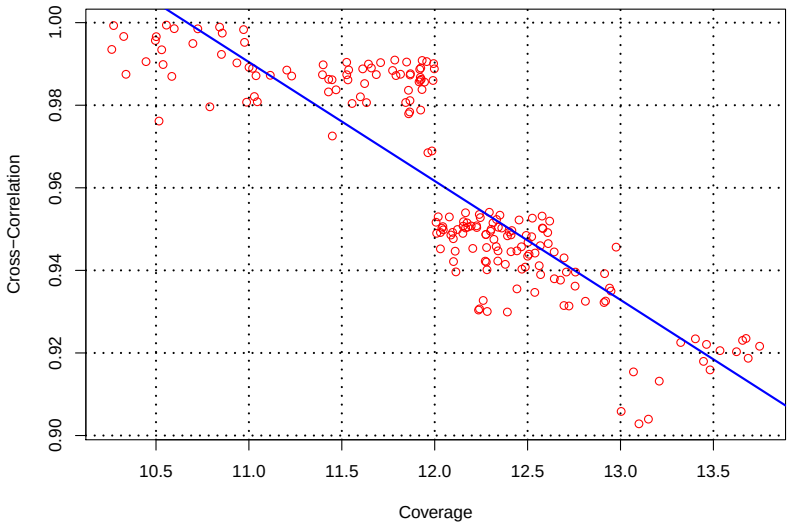


Figure 3.28: Correlation between coverage and normalized cross-correlation results

3.7 Remarks

This chapter started with a discussion of finding the homography from feature correspondences in images. The estimated homography represents a transformation from the source image to the destination image and RANSAC is a robust approach for finding a good estimate.

Later, a quantitative, robust measure was proposed to evaluate the spatial coverage of feature points in an image, able to determine whether points are aggregated at multiple scales. Based on this measure, an evaluation of a range of state-of-the-art feature detectors was performed; the evaluation method regarded the imagery and the detector as the two independent variables affecting coverage, and significance was assessed using ANOVA.

The results revealed that there is indeed statistical significance between the performances of detectors. SFOP was found to be superior to other detectors, while there are also some detectors whose performance differences were not statistically significant. These findings are broadly consistent with those obtained by other researchers using different approaches, increasing our confidence that these performance differences are real.

Experiments were also performed on stitching overlapping regions into panoramas, confirming that better coverage yields a better quality results which do not have any blurring or misalignment problems.

From the list of features evaluated in this chapter, SURF was selected as the detector of choice for the vision-based user tracking algorithm described in the following chapter. The reason behind this is that most feature detectors that yield a better coverage than SURF in Table 3.5 do not include a descriptor. The one exception to this is SIFT which is not very suitable for real-time operation.

CHAPTER 4

VISION BASED USER TRACKING

Finding the position of the user is crucial in AR applications, as the pose of the user defines the view that will be used to visualize any 3D models. Furthermore, as mentioned in section 2.9, using a sensor for both finding the position of the user and acquiring the images which will be used as the background for augmentation is a better way of utilising available resources.

From the literature it is known that pure SFM approaches cannot be used in a real-time system since they include computationally expensive methods such as bundle adjustment, which uses an iterative optimization method to refine the initial motion estimate. Such methods are not considered suitable for AR applications since on-line pose tracking is required, while those methods employ off-line, batch processing solutions [86,90]. Furthermore, SFM methods are prone to errors due to noise [78] and cannot guarantee repeated localization and robustness [75].

SLAM methods, widely used by the robotics community, were proposed as

alternatives to SFM for real-time pose tracking for AR [196, 198, 199]. These methods offer an incremental solution to the tracking problem, as new features are added to the system state when the camera is observing an area that was not previously mapped. However, this ability comes with a problem known as data association, one that must be overcome in all SLAM systems.

Ideally, each feature stored in the system corresponds to a real-world landmark. If data association is not performed properly, the system will add new features from the same landmark to the system state and unnecessarily grow the filter because the system cannot recognize these landmarks using the available representation. Growing state size reduces the accuracy of tracking since this causes linearisation problems. This problem also results in a decrease in computation speed. Hierarchical approaches [108, 116] would help here if the filter can converge and perform accurate tracking for a small area, so that this can be extended to neighbouring areas and finally covering a large area.

A second problem is due to the increasing size of the environment, as described later in the chapter. Indoor environments where SLAM methods [196, 198] work accurately require fewer features for tracking. Furthermore, the structure of indoor environments also facilitate the use of such systems: for instance, the lighting in an indoor environment does not change as much as in an outdoor environment.

Considering the promising results from studies like [69, 199, 213], this research initially started with a monocular SLAM implementation, which unfortunately could not achieve the required accuracy for tracking. Following this, a different approach was taken, using selected frames from a sequence, and this approach provided better results by incorporating ideas from the analysis of the spatial distribution of image features presented in the previous chapter.

The rest of this chapter starts with a discussion of camera calibration, which

aims to find the internal parameters of the camera that affect the imaging process. The calibration result found will be used later in the chapter to create realistic projection matrices, along with external parameters obtained using a motion estimation algorithm.

The filtering methods discussed in section 2.2 will be examined in more detail, giving a better insight of the theory and practical implementation of the two most popular types of filters, namely Gaussian (*e.g.* KF) and non-parametric (*e.g.* PF). Discussion will include how these two filters update their current states using observations from the environment they model. These filters will also be used in a sample tracking case of an imaginary user following a sinusoidal trajectory. Tracking results will be given and a comparison between the two filters will be presented.

The discussion continues with a recent implementation of the visual EKF SLAM algorithm [213,241]. Details of the algorithm will be provided in steps since it follows the same prediction–measurement–update cycle of a KF, in addition to the different representations used for features in order to achieve better tracking accuracy. Examples of how this implementation can fail in case of severe rotations will be provided, along with a discussion of why this approach could not be used in an outdoor environment.

The chapter then introduces an alternative approach, based on two-view geometry, using keyframes extracted from input images acquired by a single camera. The description of this approach starts with a statement of the assumptions and challenges due to outdoor operation and alignment of the camera, since both of these factors affect feature matching due to changes in scale and calculating depth of features. Details of this approach will be provided as steps including keyframe extraction, finding different motion estimate solutions and finally using triangu-

lation in order to decide on the final motion estimate among others. This motion estimate is used in a way similar to dead-reckoning, as will be described later in the chapter.

Finally, results will be presented for two feature descriptors on two different datasets of a user walking in an outdoor environment.

4.1 Camera Calibration

Camera calibration is an important process in almost all applications involving vision in a 3D world. Calibration means finding camera parameters which affect the imaging process in order to model this in a more realistic way than the ideal camera representation described in section 2.1.1. The camera parameters can be examined under two groups namely internal (*e.g.* focal length, distortions, *etc.*) and external parameters (*e.g.* position and orientation), described in the following.

Internal (intrinsic) parameters are directly related to the imaging process in any vision application. Focal length is represented as f and denotes the distance between the optical centre of the camera (C) and the imaging plane, with centre $c = (c_x, c_y)$, as shown in Figure 4.1.

In CCD cameras, there is possibility of having pixels that are not square [31]. For this reason, additional scale parameters (m_x and m_y) in the x and y directions respectively are used to scale the focal length: $f_x = m_x \times f$ and $f_y = m_y \times f$. s_θ is called the *skew* parameter, $s_\theta = \cot\theta$ where θ is the angle between the pixel axes (Figure 4.1). In most cameras this angle is close to 90°, so s_θ is close to 0 [25].

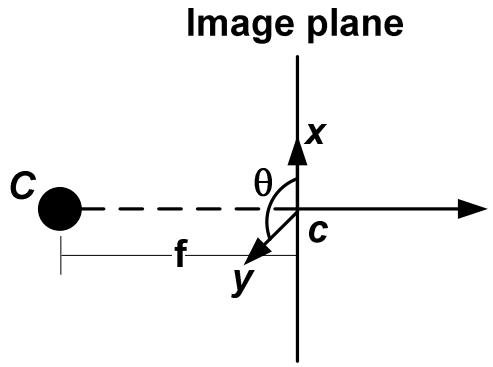


Figure 4.1: Internal camera parameters. C is the optical centre while $c = (c_x, c_y)$ is the principal point, the centre of the image plane.

Using these parameters a calibration matrix is constructed

$$K = \begin{bmatrix} f_x & s_\theta & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

This calibration matrix is applied to the camera matrix (described in section 4.4) for projecting features.

In addition to the parameters described above, there are also parameters, the *distortion coefficients*, which model the aberrations of optical lenses [242]. For instance, if the lens is not completely perpendicular to the imaging plane, the locations of the pixels will differ from their ideal values. These distortions can be in form of spherical aberration which is the problem of incoming light rays passing through different focuses due to the imperfections in the spherical shape of the lens; or coma which means that features close to the periphery of an image have a different shape to those near the centre.

In a simplified distortion model, there are two types of distortions [25, 243].

The first type is *radial distortion* which can simply be explained by lines appearing to bend near the edges of an image. *Tangential distortions* are due to alignment problems of the lens relative to the imaging plane during the manufacturing process of the camera. In this distortion, the lens is not placed perfectly parallel to the imaging plane and this imperfection manifests itself as an elliptic displacement of projected points [39].

These parameters are represented in form of a 5×1 vector kc:

$$kc = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} \tag{4.2}$$

where k_1, k_2, k_3 are the coefficients for radial distortion and p_1 and p_2 the tangential distortion parameters. These distortion parameters are mainly used in chapter 5 where projections of both RGB and depth sensors are used.

The second type of parameters, known as external parameters, define the position and orientation of the camera in world coordinates. The position is represented as a translation matrix t and the orientation as a rotation matrix R. These parameters do not directly affect the imaging process in case of a stationary camera and a static scene. However, these parameters will obviously contribute to the process for the case of a moving camera and the dynamic scene. These two external parameters are used to create a camera matrix which will be presented later in section 4.4.3 in order to calculate the projections of features according to the position and orientation parameters obtained using the keyframe based algorithm of section 4.4.

Camera calibration was performed in order to find the intrinsic distortion parameters of the simplified model using the Camera Calibration Toolbox for

Matlab [35][1]. The web-camera used for this purpose and in the rest of the thesis is a Logitech QuickCam Pro9000[2] which has a 2 megapixel sensor, a diagonal FOV of 75° and can record videos at 30fps. In the experiments, images of 640×480 pixels were captured.

For the calibration process, a rectangular grid was prepared (Figure 4.2) and attached to a cardboard measuring 18cm×27cm.

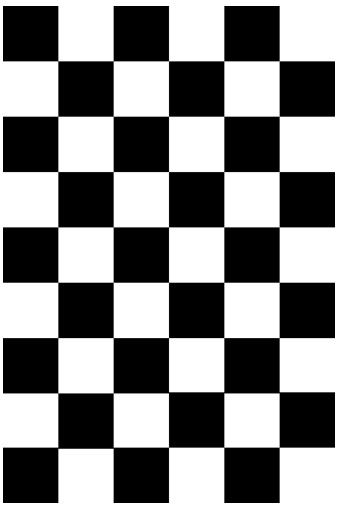


Figure 4.2: Calibration grid

Images of the calibration grid were captured. The grid was moved in the scene and tilted, since one needs to acquire images from different positions and orientations. Usually 20–25 images are found to be enough for the calibration process [35]. Figure 4.3 shows some of the images acquired for calibration.

The toolbox works semi-automatically in that the user first needs to select the outer boundary of the calibration grid of Figure 4.2 in each of the images. This information is used to estimate the positions of corners for a given size of each square (30mm). During this phase, the user can also enter an initial estimate of

[1]Available at `http://www.vision.caltech.edu/bouguetj/`

[2]`http://www.logitech.com/en-gb/support/3056`

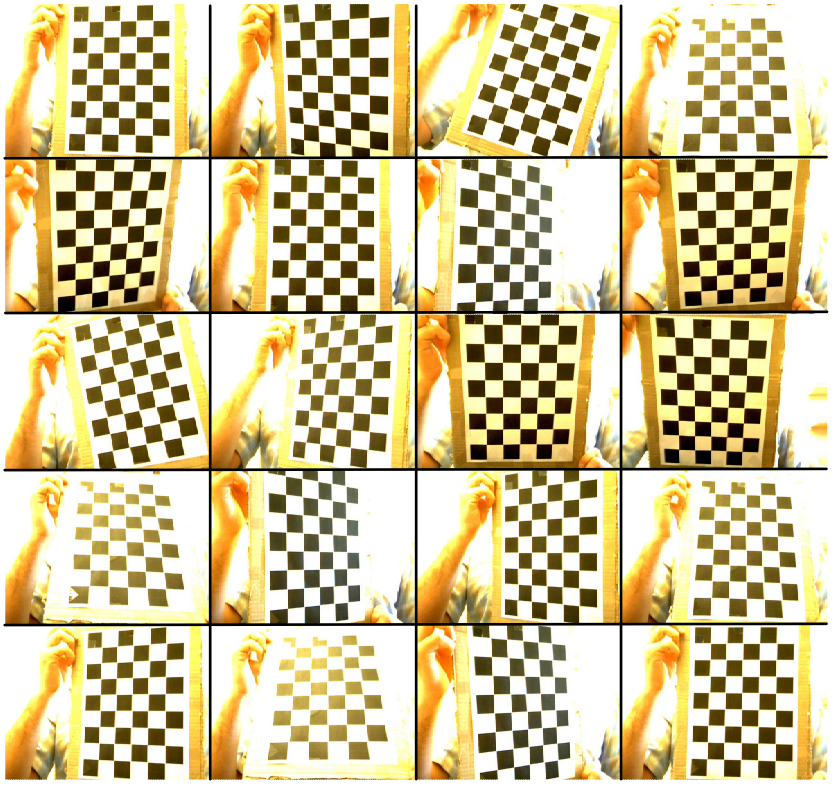


Figure 4.3: Calibration images

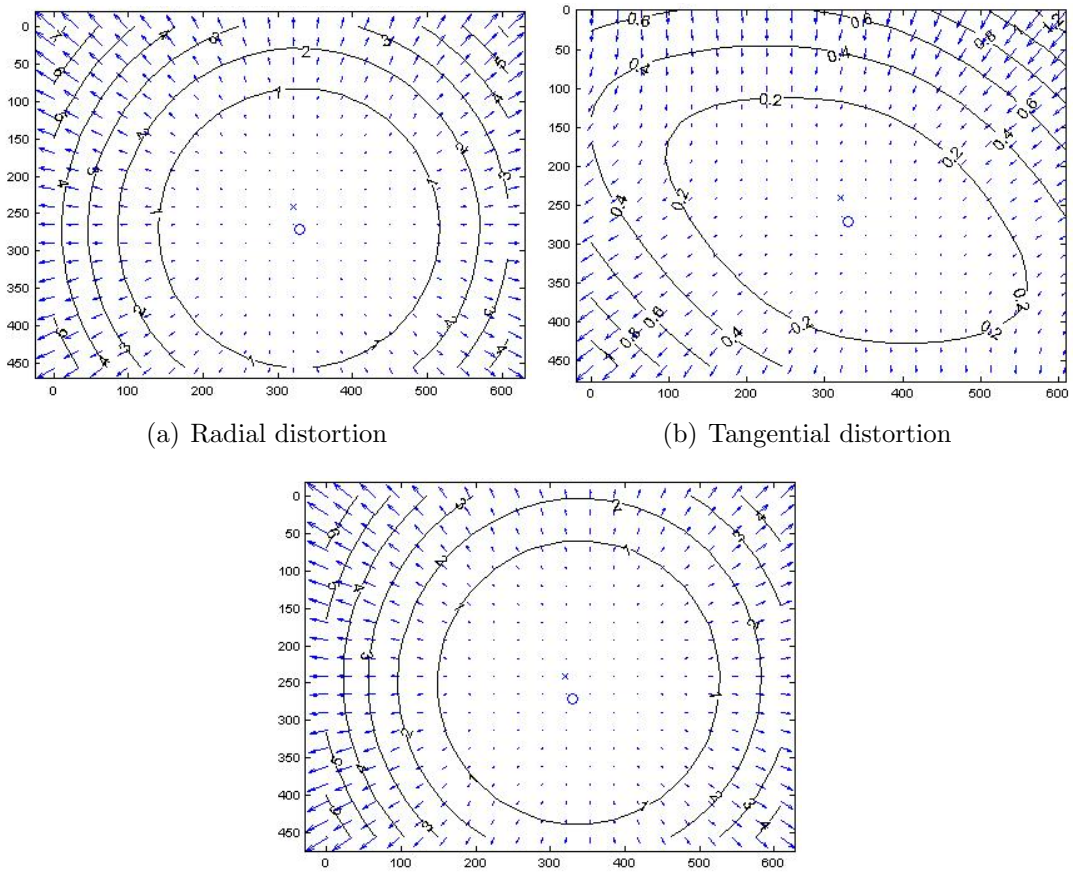
the distortion if the estimated positions for the corners are not aligned with the actual corners. Then, corner features are extracted to 0.1 pixel accuracy using this initial estimate. After this step, the toolbox first finds a closed-form solution for the calibration parameters and then performs an optimization in order to minimize re-projection errors.

The results of the calibration process are presented in Table 4.1 along with the tolerance values calculated by the toolbox. The principal point was found to be at $(328.064 \pm 6.386, 269.565 \pm 9.846)$, indicating that the estimated values can be between 321.678 and 334.45 for the x component of the principal point and between 259.719 and 279.411 for the y component. Ideally this position was expected to be exactly at $(320, 240)$ for images of 640×480 pixels. The skew parameter was found to be 0, indicating that the angle between x and y axes are exactly 90°.

Table 4.1: Calibration results for distortion parameters

	Value			**Tolerance** (±)		
	x	**y**		**x**	**y**	
Focal length	884.38	876.373		6.916	7.019	
Principal point	328.064	269.565		6.386	9.846	
Skew		0			0	
	k1	**k2**	**k3**	**k1**	**k2**	**k3**
Radial coefficients	0.1263	-0.2237	0	0.04466	0.4288	0
	p1	**p2**		**p1**	**p2**	
Tangential coefficients	0.002517	-0.0012		0.00519	0.003302	

Figure 4.4 provides visualizations of the effect of the distortions on the pixel locations. The arrows show the direction of displacement, while the numbers on the arrows show the amount of displacement of pixels. These visualisations help decide on the distortion model to be used for feature searches in the images in the actual SLAM implementation described later in the thesis.



(a) Radial distortion

(b) Tangential distortion

(c) Complete distortion

Figure 4.4: Visualizations of the distortion parameters, obtained using [35]

By looking at the visualization of the distortions, it can be deduced that the effect of the tangential distortion is not significant for the camera used in the tests. In addition, the displacements of pixels in the radial distortion were deemed to be not significant as they do not exceed 6 pixels: the reason for this is the computation time required for undistorting an image. In any case, this is compensated for by the search regions described in section 4.3.2.

4.2 Filtering Methods

An algorithmic model can be considered as an approximation to the real world it is modelling. There can be some factors that cannot be directly incorporated into a model because they cannot always be predicted at the time of modelling, as described earlier in section 2.2. These unpredictable factors may adversely affect the system performance [244]. Filtering methods are used to handle the unpredictable factors of a process or a system. For a system trying to track the position of a person, these can arise from a variety of sources [94, 120].

First, it is not easy for the tracking system to predict the motion pattern (walking slowly, fast, standing, turning, *etc.*) which the user is following in advance. If this was known it could be given as an initial estimate for a filter which is investigated in chapter 6. For this reason, tracking a person is more challenging than tracking a robot which can be given motion commands and are almost guaranteed to follow them except for mechanical limitations such as wheel slip [245]. Even if one assumes that people can also follow motion commands given to them, then the movement of two different users will be different since the motion cannot be easily replicated by different users.

A second important source of uncertainty is due to sensor limitations, or static

errors as mentioned in section 2.8. For instance, a GPS sensor is not perfectly accurate in terms of the positional information it provides and, what is worse, the accuracy is affected by the visibility of satellites. Similarly, for a camera the resolution of the images acquired by it is limited; and visual noise can be a problem in the detection of image features from real world objects in a repeatable way.

Finally, errors in the system model arise because of algorithmic approximations. If the phenomenon is not modelled perfectly, then it will not be surprising when the approximation fails to converge. Certainly, this depends on the nature of the phenomenon. If the initial model is failing because the phenomenon is constantly changing, then an adaptive model may be required (see chapter 6). Insufficient tuning of parameters may cause additional levels of uncertainties in the system, a problem also related to the modelling.

The systems proposed in this thesis make use of two types of filters, namely Gaussian (*e.g.* KF) and non-parametric filters (PF). These two types are explained using an example case of simple location tracking. Let us consider a person walking along a sinusoidally-shaped path defined by

$$y = 4 \sin x \tag{4.3}$$

To model the person's path, two components of the position in pixel coordinates need to be stored in a *state*, denoted $\mathbf{x}$, where

$$\mathbf{x_i} = (x_i, y_i)^T \tag{4.4}$$

Note that the position of the person is modelled as a point, so the orientation is not a considered in this example.

State $\mathbf{x}$ is where the system believes the person is at time t_i. An initial position may or may not be available in practice, though it is usually considered that this position is the one when the tracking started. For this example, the person is considered to start walking from a known position.

It is also assumed that the person informs the system about his/her position (as x and y coordinates in pixels) using an accurate sensor (similar to a differential GPS) at regular intervals at a frequency high enough to perform tracking accurately. Now, two different filtering algorithms (a KF and a PF) will be used to track the position of the person.

4.2.1 Kalman filtering

The KF is a Gaussian filter used to model a continuous linear system recursively with process uncertainty and noise [246]. A state denotes the condition of the system at a given time. As the system is continuous, there will be many states in the system and one state will move into the next state with a transition function.

The transition function predicts the next state $\hat{\mathbf{x}}$ based on the current state $\mathbf{x}$. As noise is unavoidable in a real process, measurements must be taken in order to verify and, if necessary, correct this prediction. The whole system can be modelled in this manner and these methods can be applied to processes such as control systems or tracking applications [247]. This cycle of prediction, measurement and update is illustrated in Figure 4.5.

For the problem of tracking the person following a sinusoidal path of (4.3), the transition function must be defined. Before proceeding, two additional parameters are added to the state in order to model the velocities as well as positions so the

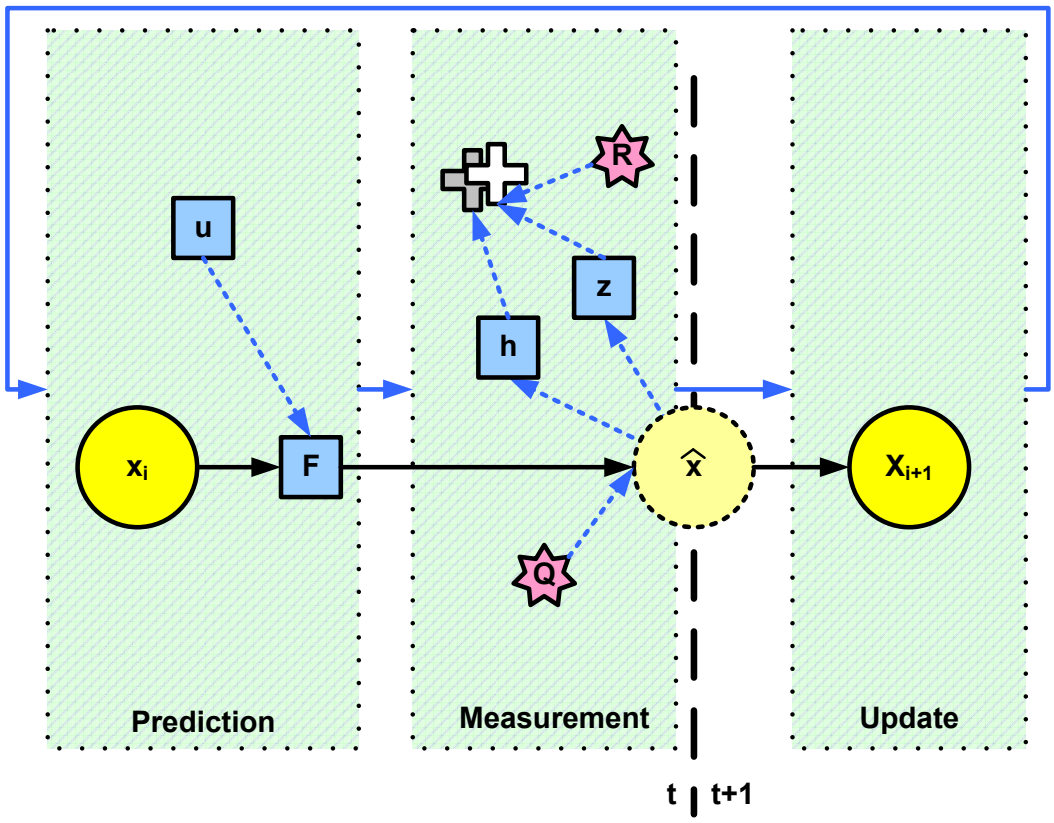


Figure 4.5: Prediction-measurement-update cycle of the KF following [117], [248] and [105]. A transition function F is applied to the state (yellow circle) in order to predict the next state shown with the dashed circle along with the process noise Q. The control input u can also be used at this stage. At the measurement phase, a measurement prediction h is generated. This prediction (shaded cross) is an idea of the system about what the measurement will be and may be different from the actual measurement result shown with white cross and denoted by z. A measurement noise R may also be incorporated when the actual measurement will be taken. Next step is updating the state with the measurement so the system will be ready for the next iteration of the cycle.

new state will be

$$\mathbf{x} = (x, y, v_x, v_y)^T \tag{4.5}$$

The tracking system does not have an idea of what kind of a trajectory the person is following, so it uses a simple transition function denoted as F with Δt being the time passed between two predictions (taken as 1 for this example):

$$F = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

Giving motion commands to the person as discussed in the previous section is not possible in this case so the control input u will not be used. However, the process noise Q cannot be avoided so these are incorporated into the prediction as shown in Figure 4.5.

For sake of simplicity, predicted measurements h are taken as the coordinates of the person after the prediction phase and the noise level for the actual measurements z is low due to the accurate sensor used. The process of KF can now be described as in Algorithm 1.

The variables used in the KF algorithm have been explained with the exception of two important ones: Σ and K. The uncertainty is handled by the system using the state covariance Σ. In other words, this variable stores the amount of confidence the state vector has about the system status. Σ can also be used to visualize this confidence by using error ellipses or ellipsoids as described by [241]. The second variable K is the Kalman gain, which can be described as the weight of the measurements taken at time t_i in the calculation of the next state x_{i+1}.

Algorithm 1 Kalman filter

Require: x_i: current state, Σ_i: current state covariance, z_i: measurement

// Prediction

$\hat{x}_{i+1} = Fx_i$

$\hat{\Sigma}_{i+1} = F\Sigma_i F^T + Q_i$

// Measurement

Observation z_i is taken as position

// Update

$K_i = \hat{\Sigma}_i h_i^T (h\hat{\Sigma}_i h_i^T + Q_i)^{-1}$

$x_{i+1} = \hat{x}_{i+1} + K_i(z_i - h_i\hat{x}_{i+1})$

$\Sigma_{i+1} = (I - K_i h_i)\hat{\Sigma}_i$

Having described the KF algorithm with all its parameters, Figure 4.6 shows the tracking result for the person moving with the trajectory of (4.3).

From the tracking result, it can be seen that the filter makes incorrect predictions initially — the green dots do not follow the red circles. However, once the filter converges, tracking performance becomes stable and continues until the end of the motion.

4.2.2 Particle filtering

A PF also performs state estimation as a KF. The difference is that a KF represents the posterior using a fixed functional form [94] whereas a PF uses a representation with particles which are independent random variables [92] uniformly distributed in the state space. This distribution allows representing a larger set of distributions, not only Gaussian, and hence having a greater advantage for non-linear motion types. This power of particle filtering comes with a trade-off, as mentioned in section 2.2.2.

For the problem of tracking a person positioned at $\mathbf{x} = (x, y)^T$, the initial

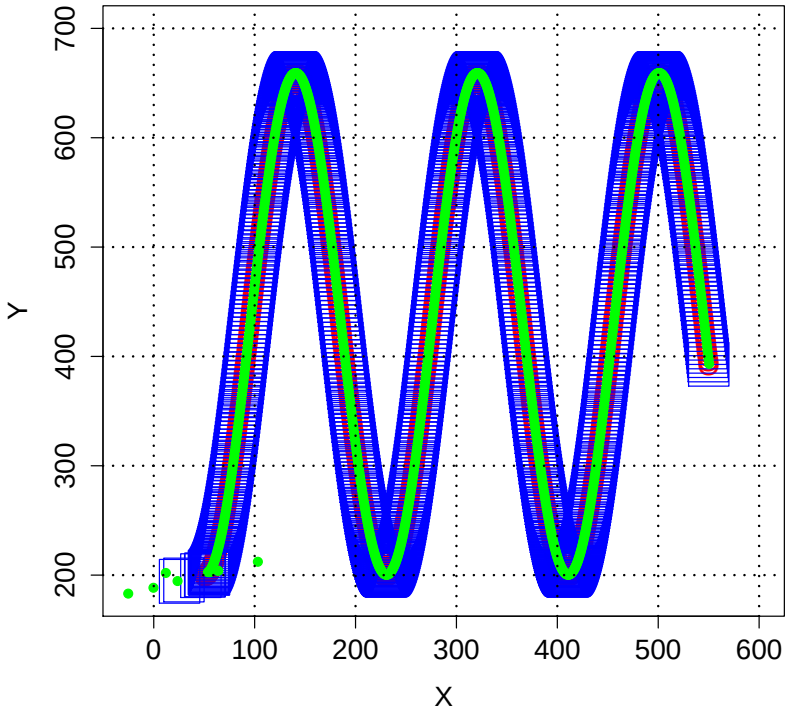


Figure 4.6: Tracking a person with KF. Red circles show the actual trajectory of the person. Green dots indicate the predicted position and blue rectangles show the state of the KF after the update.

state is set using this known position and the particles scattered within a 50-pixel radius of that point. The state $\mathbf{x}$ is then updated using two models, one dynamic and the other observational.

The dynamic model A generates the predicted (hypothetical) state $\hat{\mathbf{x}}$ and is initialized to the identity matrix for simplicity:

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.7}$$

At each frame the predicted state is calculated for all particles $(1 \dots n)$:

$$\hat{\mathbf{x}}_i = A\mathbf{x}_i \tag{4.8}$$

Particle weights are updated using the observational model. Measurements ($m = (m_x, m_y)$) are used to refine the particle confidences using Gaussian sampling as

$$\begin{aligned} w_{i_x} &= e^{\frac{-1}{2\sigma_x^2}(m_x - p_{i_x})^2} \\ w_{i_y} &= e^{\frac{-1}{2\sigma_y^2}\left(m_y - p_{i_y}\right)^2} \\ w_i &= w_{i_x} \times w_{i_y} \end{aligned} \tag{4.9}$$

where w_i denotes the weight of particle i and σ_x^2 and σ_y^2 are the variances of the samples for the x and y coordinates of the position. The hypothetical values of the person's position are stored in p_{i_x} and p_{i_y}. This process is followed by re-sampling, in which the updated confidences will be used to create a new set of particles of the same size [39, 94].

Figure 4.7 shows the tracking result for the walking person following the sinusoidal trajectory of (4.3) using the *Condensation* algorithm. The experiment was

run with 200 particles with a search region of 50×50 pixels for a total of 1000 frames.

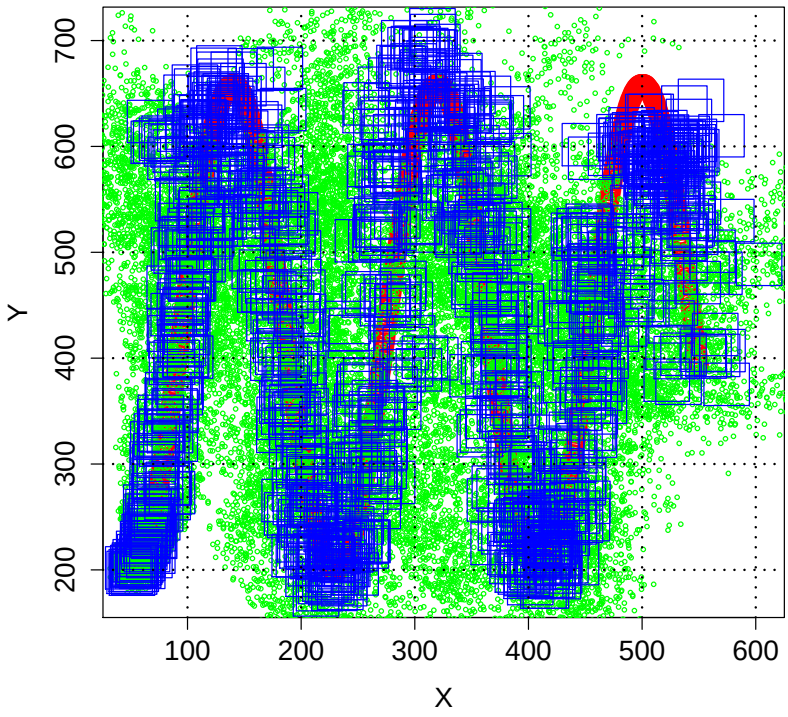
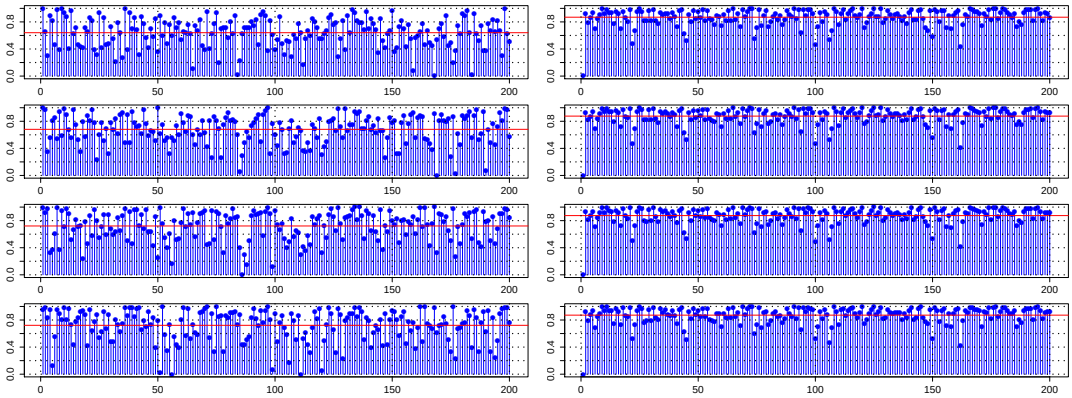


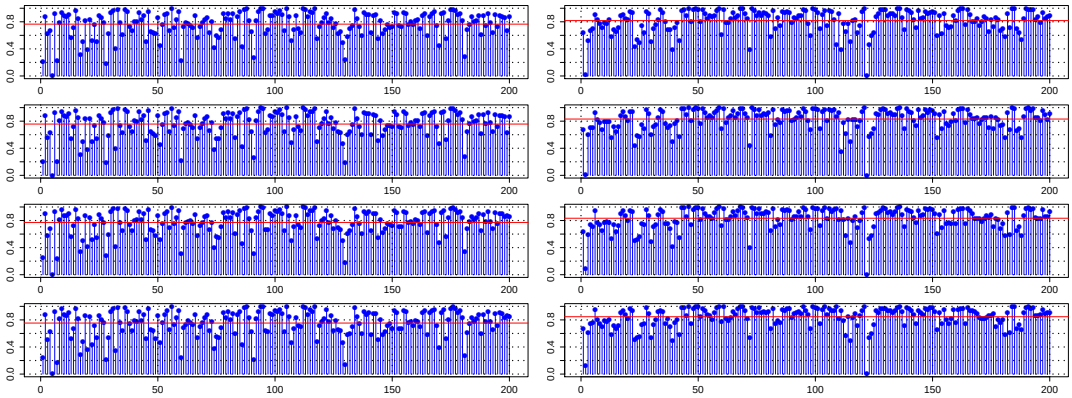
Figure 4.7: Tracking a walking person with PF. Red circles show the actual trajectory of the person. Hypothetical positions suggested by the samples are shown with green circles and blue rectangles indicate the position suggested by the sample with the highest confidence. The estimation is performed using 200 particles from which 25 are selected based on confidence levels and displayed.

The changes of particle confidences are depicted in Figure 4.8. The red line, showing the mean particle confidence, moves upwards on frames where the measurements support the hypothetical positions proposed by each sample; and downwards when the estimate is not correct.



(a) Frames 1 (top) to 4

(b) Frames 249 (top) to 252



(c) Frames 501 (top) to 504

(d) Frames 749 (top) to 752

Figure 4.8: Change in particle confidences for different frames, red line indicates the mean confidence.

The changes in confidence levels are further analysed in Figure 4.9. This analysis reveals that the frames where the mean confidence level decreased significantly correspond to when the person changes his/her direction of motion *i.e.* where the sinusoid has its peaks.

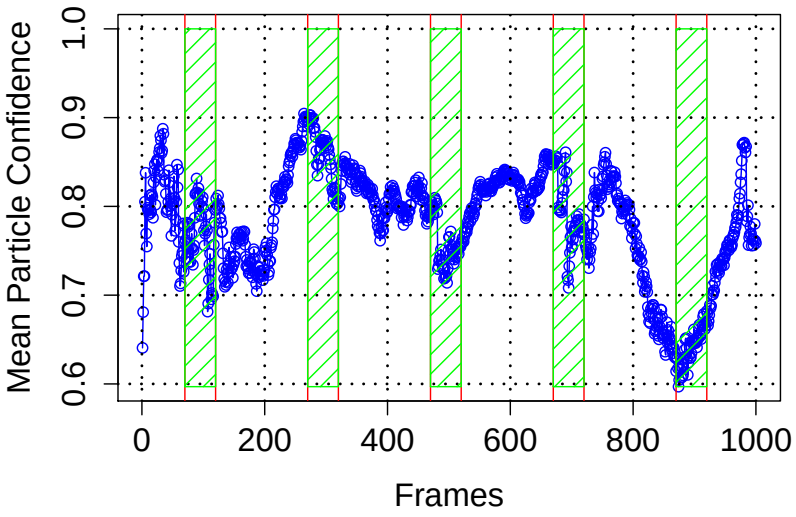


Figure 4.9: Mean values for particle confidences throughout 1000 frames. The shaded regions show when the confidences decrease significantly due to the change in direction at the edge locations of the trajectory.

4.2.3 Comparison

The two examples of filtering explained in sections 4.2.1 and 4.2.2 have shown that KF and PF can be used to follow a trajectory. Each filter has its own strengths and weaknesses depending on the application, and the idea here was not to praise one filter over the other but to provide an example of tracking. In this particular example, the results from the KF is better than the one from PF mainly because of the small number of particles. Experiments with 500 particles produced better results, though using so many particles significantly reduced the speed.

Following the discussions in sections 2.2 and 4.2, a comparison for these two types of filters is given in Table 4.2.

Table 4.2: Comparison between KF and PF

	Weaknesses	**Strengths**
KF	• Linear inputs and outputs • Filter may not converge if error is not Gaussian.	• Fast filter • Perfect estimator if assumptions are satisfied.
PF	• Accuracy may suffer due to simple modelling. • Computationally expensive • Trade off between accuracy and computational cost	• Ability to model non-linear transformations • Non-parametric • Simple model

4.3 A Monocular SLAM Implementation Using EKF

The design of the vision-based part of the complete tracking system of chapter 6 was initially started by implementing a monocular SLAM system using an EKF, following the approaches of Davison [69] and Civera [213]. The choice here was mainly due to the performance of this algorithm in different applications which were mainly indoor environments.

An alternative approach was the one proposed by Mouragnon [85], which uses the SFM methods (*i.e.* PnP together with bundle adjustment) described in 2.1.3, which was tested on an autonomous car. The latter approach was not chosen for this research since the intended system would be tracking a user, which is more challenging than tracking a car due to the first source of uncertainty mentioned in section 4.2. Furthermore, the system demonstrated in this thesis requires time for not only tracking the user but also generating graphics in order to prepare the output for augmentation.

The following sections provide details of the main design structures and the principles underlying this implementation and will provide tracking results for a

hand-held camera.

4.3.1 Complete state model

The complete state vector $\mathbf{x}$ consists of two main parts. The first part is the camera state and the second part is reserved for features. This is represented as

$$\mathbf{x}_t = \begin{pmatrix} \mathbf{x}_v, & f_1, & f_2, & \ldots, & f_n \end{pmatrix}^T \tag{4.10}$$

The camera state storing the position and orientation of the camera is represented with $\mathbf{x}_v$. The rest of the state vector is composed of features observed from the environment, each represented with f_i.

Camera state

The part of the complete state model representing the camera is called the camera state (a.k.a. vehicle state) and comprises several parameters

$$\mathbf{x}_v = \begin{pmatrix} \mathbf{r}^{WC}, & \mathbf{q}^{WC}, & \mathbf{v}^W, & \omega^C \end{pmatrix}^T \tag{4.11}$$

where $\mathbf{r}^{WC} = \begin{pmatrix} x_c, & y_c, & z_c \end{pmatrix}^T$ is the 3D position of the centre of the camera (the optical centre) in the world coordinate system. $\mathbf{q}^{WC}$ is a quaternion used to store the orientation of camera with respect to the world frame. Quaternions are used since an orientation can be represented with only four parameters and so will not add many parameters to the state hence will not add to computation. $\mathbf{v}^W$ stores the linear velocities of the camera in 3 axes and ω^C stores the angular velocities, represented in the camera coordinate system. The complete camera state comprises 13 components.

Feature representation

The features in the environment are shown as f_i in the second part of the complete state vector. Each feature here corresponds to a 3D feature in the environment through which the camera is moving. The only sensor here is a camera and these features are observed using this. Features are represented in one of two possible methods: inverse depth coordinates $\mathbf{y_i}$ and Cartesian coordinates $\mathbf{x_i}$; these are described below.

The inverse-depth representation is based on [249] where using inverse depths was proposed in order to improve accuracy in depth estimation. The method presents an undelayed initialization and can handle features which are far from the camera, presenting little parallax (see section 4.4 and Figure 4.14 for a more detailed discussion on this). The infinity of a point is defined as the distance where the camera cannot observe the parallax although it is moving relative to the feature [213].

This representation shown in (4.12) models a feature point with 6 parameters including the camera location when the feature is observed for the first time, azimuth and elevation of the ray from the centre of the camera to the feature and the inverse depth parameter.

$$\mathbf{y}_i = \begin{pmatrix} x_{c,i} & y_{c,i} & z_{c,i} & \theta_i & \phi_i & \rho_i \end{pmatrix}^T \tag{4.12}$$

where $x_{c,i}$, $y_{c,i}$ and $z_{c,i}$ show the 3D camera coordinates at the time when the feature was first observed. θ_i, ϕ_i and ρ_i represent the azimuth, elevation and the inverse depth of the feature f_i in the camera coordinates respectively.

Using the inverse depth representation, features can be initialized with a single image observation, meaning that they can be used to estimate the camera position

and orientation. For the initial frames, the feature does not contribute much in estimating the position of the camera but it is helpful in estimating the orientation. This initial hypothetical feature information will be improved as the camera moves in the scene and observes better parallax from the feature. The actual point feature represented with (4.12), can be shown as follows:

$$\mathbf{x}_i = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} = \begin{pmatrix} x_{c,i} \\ y_{c,i} \\ z_{c,i} \end{pmatrix} + \frac{1}{\rho_i} m(\theta_i, \phi_i) \tag{4.13}$$

where $m(\theta_i, \phi_i)$ is a unit vector extending from the camera's optical centre to the feature and defined as

$$m\left(\theta_i, \phi_i\right) = \begin{pmatrix} \sin\theta_i \cos\phi_i, & -\sin\phi_i, & \cos\theta_i \cos\phi_i \end{pmatrix}^T \tag{4.14}$$

As mentioned above, the inverse depth representation is used when a feature is observed by the camera for the first time and this initial representation is converted to the second representation of classical Cartesian coordinates, where the position of each feature is represented as

$$\mathbf{x}_i = \begin{pmatrix} x_i & y_i & z_i \end{pmatrix}^T \tag{4.15}$$

This conversion from the inverse depth to the Cartesian representation takes place after a feature has been observed by the camera multiple times (*i.e.* until the depth of the feature can be estimated with some confidence level [69]). This conversion reduces both the size of the state and the state covariance since the feature is now stored using 3 parameters rather than 6.

4.3.2 EKF SLAM phases

The complete EKF SLAM algorithm can be broken down to 3 main phases: the prediction phase, the measurement phase and the update phase, as in Algorithm 1. These phases are repeated for each frame captured from the camera. This subsection presents these phases.

Prediction

Prediction corresponds to the first two lines Algorithm 1. This phase starts with the estimation of $\bar{\mu}_{t+1}$ using the transition function $f_v(\mu_t)$ for the camera. Note that the control input u_t is not actually used in f_v in (4.16) since it is not required in monocular SLAM.

$$f_v(\mu_t) = \begin{pmatrix} \mathbf{r}_{t+1}^{WC} \\ \mathbf{q}_{t+1}^{WC} \\ \mathbf{v}_{t+1}^{W} \\ \omega_{t+1}^{C} \end{pmatrix} = \begin{pmatrix} \mathbf{r}_{t}^{WC} + \mathbf{v}_{t}^{W}\Delta t \\ \mathbf{q}_{t}^{WC} \times \omega_{t}^{C}\Delta t_{quaternion} \\ \mathbf{v}_{t}^{W} + \mathbf{V}^{W} \\ \omega_{t}^{C} + \mathbf{\Omega}^{C} \end{pmatrix} \tag{4.16}$$

Δt is the time difference between two calls to the prediction function. The 3D coordinates of the camera are updated according to the physical law of *new position* = *current position* + *speed* × *elapsed time*. The new position is calculated using linear velocities. For the orientation of the camera, the same logic applies, though it is more involved as the orientation is stored as a quaternion. $\mathbf{V}^W$ and $\mathbf{\Omega}^C$ represent the noise parameters for the linear and angular velocities respectively. These parameters are used to obtain better estimates for the camera motion, which uses a constant velocity model.

As the features in the environment are considered to be static, they are not

affected by the prediction therefore the main transition function can be given as

$$f(\mu_t) = \begin{pmatrix} f_v(\mu_t) \\ 0_{f_1} \\ 0_{f_2} \\ \vdots \\ 0_{f_n} \end{pmatrix} \tag{4.17}$$

Measurement

The measurement phase tests the validity of the predicted position from the previous phase. First, a visibility test is applied to each feature in order to decide which features can be observed for measurement. This test is based on the current (*predicted*) camera position and orientation estimate. For features with an inverse depth representation, a directional vector pointing to the feature is calculated and projected onto image coordinates, whereas features with the Cartesian representation are directly projected onto image coordinates.

An innovation covariance S, which indicates the uncertainty in the measurement, is computed for each *visible* feature f_i in order to define a search region for the feature using the measurement prediction h and measurement noise R, shown in Figure 4.5. This is accomplished using

$$S_{t,i} = h_{t,i}\bar{\Sigma}_t h_{t,i}^T + R_t \tag{4.18}$$

where

$$R_t = \sigma_R^2 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{4.19}$$

The resulting $S_{t,i}$ defines an elliptical search region around the predicted coordi-

nates of the feature in the image [241]. This ellipse is defined as follows: the centre coordinates of the ellipse is the measurement prediction h. The region defined by the ellipse with 2σ that corresponds to 95.5% confidence level (3σ can also be used for a confidence level of 99.7% but the increased region size will increase the computation time when searching for a feature.). The axes of the ellipse along x and y coordinate axes are

$$\begin{matrix} axis_x \\ axis_y \end{matrix} = \begin{matrix} 2 \times \sqrt{\sigma_{0,0}} \\ 2 \times \sqrt{\sigma_{1,1}} \end{matrix} \tag{4.20}$$

where $\sigma_{i,j}$ corresponds to the i^{th} row and j^{th} column of the S matrix. The orientation of the ellipse, θ, is calculated as

$$\theta = \frac{1}{2} \tan^{-1} \left(\frac{2\sigma_{0,1}}{\sigma_{1,1} - \sigma_{0,0}} \right) \tag{4.21}$$

The ellipse defined here can be used to display the search region as an ellipse on images (or an ellipsoid in a 3D view); however, $axis_x$ and $axis_y$ are used to define the boundaries of the actual search rectangle in the image as this is required for finding feature matches. These feature matches can be found using NCC on the patches surrounding the features or by comparing descriptors as discussed in section 2.1.2.

Update

This part constitutes the last three lines of the Algorithm 1 and is easier than the first two phases as the system has all the necessary information at this stage to perform the update on the state and its covariance. It is also important to note that only the features that can produce a successful match are used to update the

filter since only they can provide useful information. Following this phase, the whole operation will be repeated for the next frame acquired by the camera.

4.3.3 Results for EKF SLAM

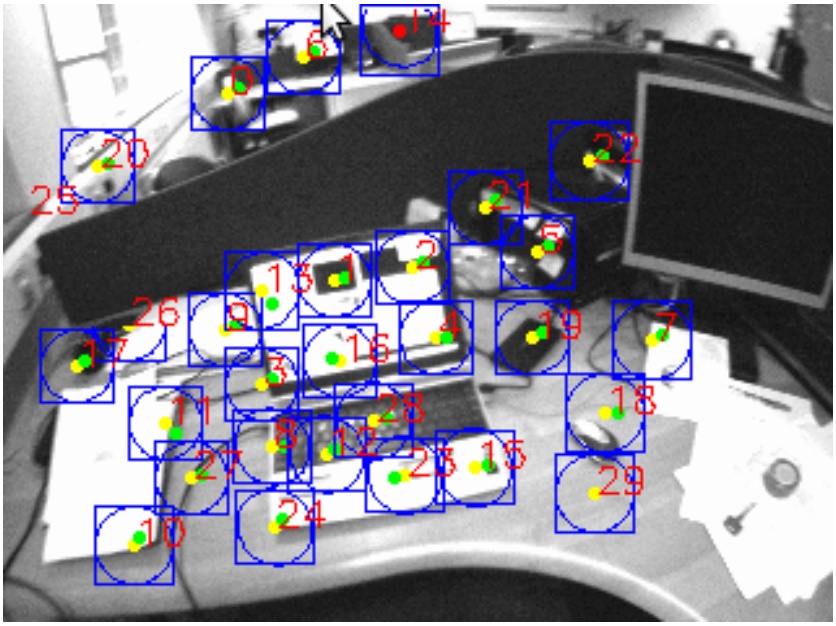
Having described the visual monocular SLAM algorithm, this section presents results for the initial implementation. Figure 4.10 presents a view of the visual tracking of features. Each feature is surrounded with a search region indicating possible locations at which the feature may reside. Selection of features were carried out using GFT described in section 2.1.2. It can be seen that some features can be successfully matched while others fail to produce a correspondence.

3D views of the tracked features are depicted in Figure 4.11. The sizes of the yellow ellipsoids indicate the amount of uncertainty present in feature positions. Notice how these uncertainties decrease as more successful measurements are taken from features while the camera moves. Successful feature measurements also play an important role in decreasing the size of the filter state since features stored in inverse-depth representation are converted into the Cartesian system after their depths are estimated with some confidence, as described in section 4.3.1.

After several frames, tracking tends to lose accuracy because of the camera's sudden and erratic (*e.g.* upside down) motion and the system starts to make incorrect predictions. The problem deteriorates when a wrong match is found in an incorrectly predicted location due to the simple descriptor used. This leads to a case in which the camera's predicted position (even it is incorrect!) is accepted because the features have resulted in a 'successful' match as shown in Figure 4.12.

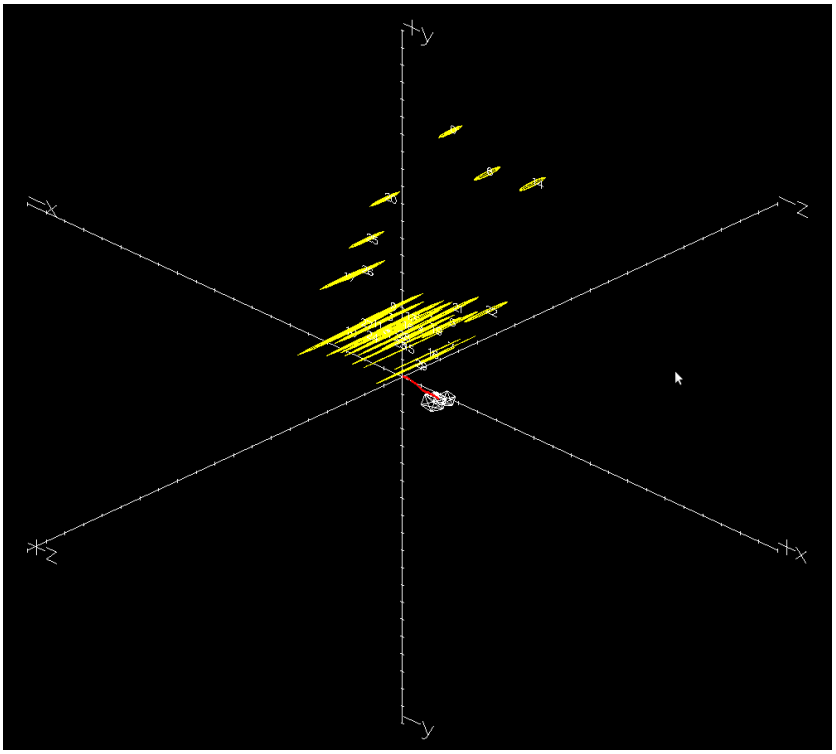
The problems occurring due to incorrect feature position prediction and spurious measurements obtained resulted in serious tracking failure which can be

(a) Prediction and matching of features

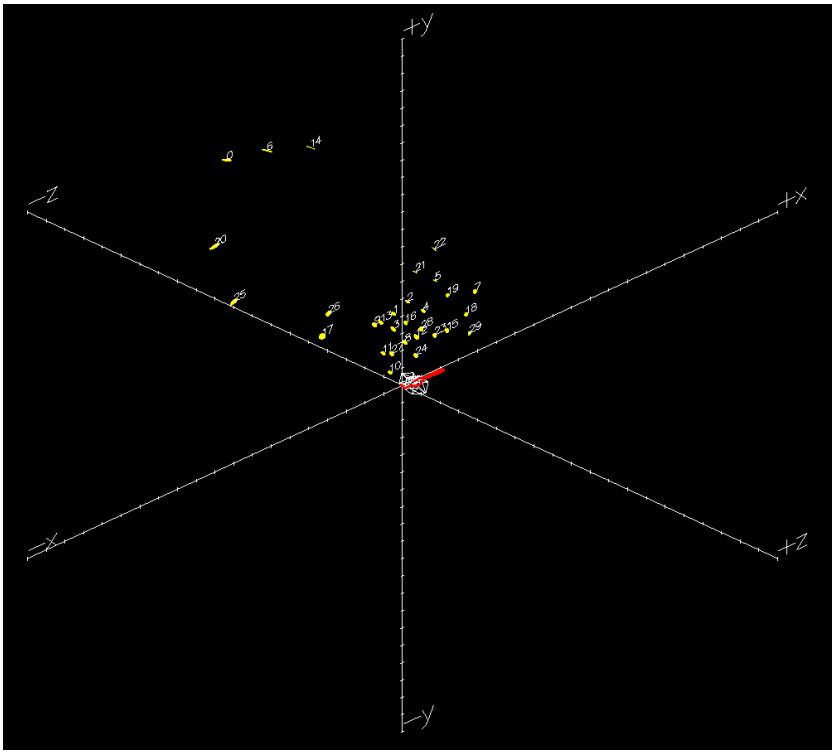


(b) Feature #14 is failing to produce a successful match

Figure 4.10: Selected features for tracking. Blue rectangles indicate the search regions around features described in Section 4.3.2. Yellow circles show the predicted locations of features and green circles show successful matches. Unsuccessful matches are shown in red circles. Dataset from [213]



(a) Ellipsoids are relatively large at the beginning.



(b) As the camera moves and takes more successful matches, positional uncertainty for both the camera and the features are decreasing, hence the size of the ellipsoids.

Figure 4.11: 3D view of camera path and features. Yellow ellipsoids show the features' locational uncertainties. The camera trajectory is shown by the red line.

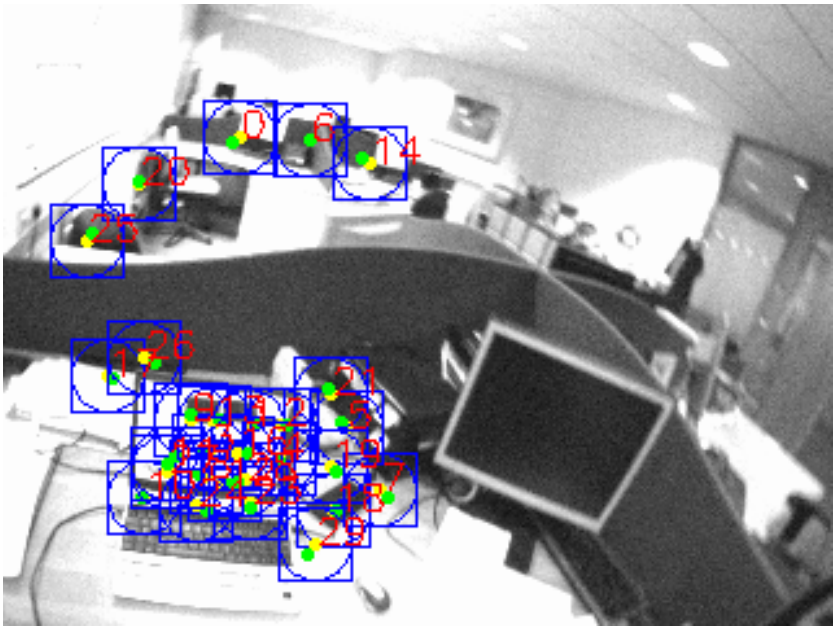


Figure 4.12: Incorrect data association. Notice where the initial position of Feature #14 (as well as other features) in Figure 4.10(b) was and where it is being predicted and matched in later frames.

described as follows:

- Data association (see section 2.3.1 and Figure 4.12) became a problem even if scale- and rotation-invariant feature descriptors (*e.g.* SURF) or two-level matching strategies (*i.e.* template matching first and then matching a BRIEF descriptor) [9] were used. The severe rotations of the camera could not be handled, even by these rotation-invariant descriptors.

- As a result of the previous problem, the state size grew unnecessarily large during execution. The reason behind this was that the available features in the map could not produce successful matches, reducing the quality of the features. After several unsuccessful matches, features started to be deleted and new features had to be extracted, eventually decreasing the tracking performance in later frames.

- It is known that large state size increases the non-linearity of the EKF [250, 251], which performs a linearisation of input and output parameters as described in section 2.2.1. Newly added features (due to matching failures of existing features and addition of these into the state as new ones) unnecessarily increased the size of the state. This resulted in a decrease in tracking accuracy.

- Tests performed in outdoor environments did not produce accurate tracking results since a larger environment requires more features to achieve tracking. A sub-mapping approach (*e.g.* [116]) was not attempted because the tracking problems were becoming serious soon after execution started, with the filter failing to converge.

Although these problems were attacked using several methods, which included using different feature detectors/descriptors, selecting features that are widely scattered across the image (chapter 3 gives a detailed evaluation on how the distribution of features affects the calculated homography) to calculate homographies, *etc.*, the implementation could not produce reliable results.

4.4 Keyframe Based Motion Estimation

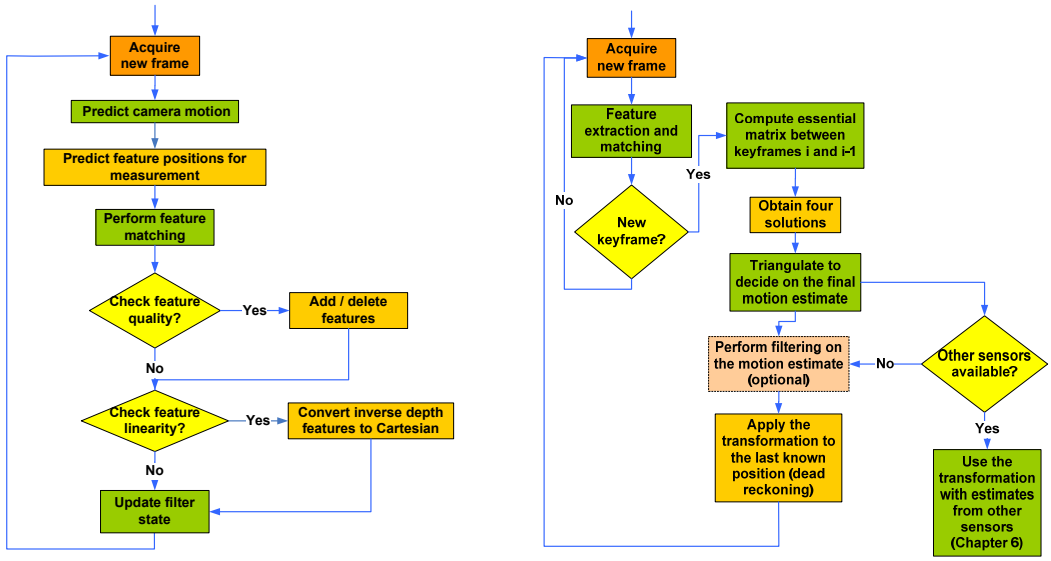
Better visual tracking results were achieved by taking a slightly different approach from the EKF SLAM system presented in the previous section. The new approach used keyframes to estimate the camera motion using two-view geometry, using a combination of the methods proposed in [31, 252] and [85, 253]. Geiger *et al.* [254] proposed dense 3D reconstruction and visual odometry, using the approach described in [31, 252], for stereo motion estimation and tracking autonomous cars. This approach also used a KF to refine the initial estimate obtained from the stereo approach for noise removal.

Instead of taking a direction that uses dense reconstruction as in [254], the method described here uses keyframes selected following the approach of [85, 253] where Royer *et al.* used PnP on features detected from the keyframes. The approach used in this chapter is slightly modified since the original method can skip too many frames, which would allow the system to miss some important movements.

The method proposed in this chapter uses keyframes acquired from a single camera aligned parallel to the direction of motion (as detailed in section 4.4.1). Features extracted from these keyframes are used to estimate the motion of the camera using two-view geometry [31]. This motion estimate was applied to the previous position in order to obtain the new position estimate, a method similar to dead-reckoning [255] where the current position of the user is deduced based on the previous position and the motion in between.

Following the approach of [254], a KF was used to refine the visual estimate initially; however this filtering part was removed later since the vision system was to be used in conjunction with other sensors (*i.e.* GPS and IMU) as described

in chapter 6. Figure 4.13 shows block diagrams outlining the processing steps for both algorithms.



(a) EKF SLAM (b) Keyframe-based motion estimation

Figure 4.13: EKF SLAM and keyframe-based motion estimation

Details of the motion estimation method will be provided in the following sections after presenting the assumptions behind the system and an important challenge due to the alignment of the camera in the following subsection.

4.4.1 Assumptions and challenges

Similar to other vision-based systems, the motion estimation method presented here depends on the following assumptions:

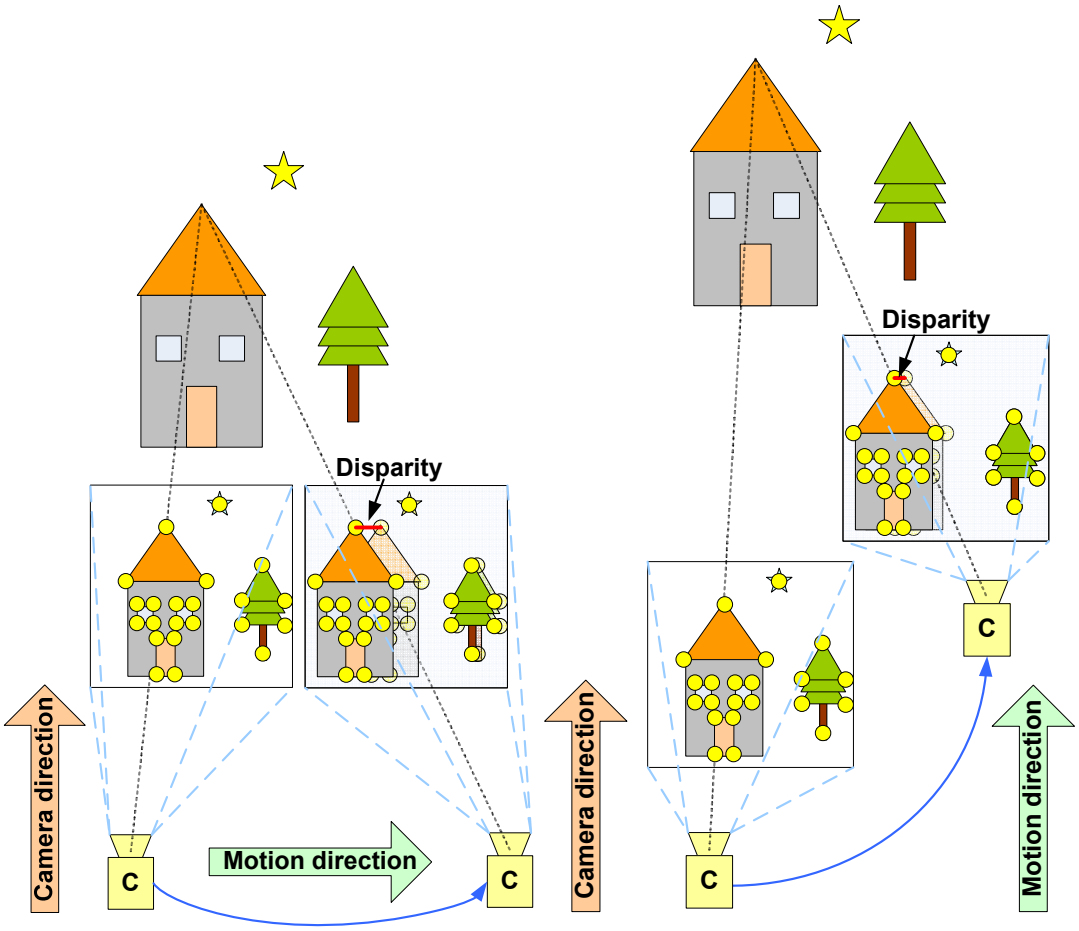
- Calibrated camera: The intrinsic parameters of the camera (section 4.1) must be known since projections will use these parameters in order to triangulate features for motion estimation.

- Static environment: Objects in the surroundings are considered to be non-moving which is different from the work in [256] where tracking was performed on moving objects.

- Known starting point: The sequence starts from a known location (can be arbitrary, but should be defined as the starting point) due to the successive nature of finding position estimates (*i.e.* dead-reckoning). This does not imply that a preprocessing step is required for the algorithm to run.

These assumptions describe an environment in which more robust results can be obtained. Before the keyframe based localization algorithm is explained, it is important to describe two challenges that were faced during the experiments due to the alignment of the camera according to the direction of motion, since this affects the accuracy of the estimation. When the camera is aligned perpendicular to the motion direction (*e.g.* the SLAM system in [257]), the disparity of features in the environment can be better observed, allowing the reconstruction to be more accurate with smaller projection errors; as a result, the estimation of camera motion will be better. Furthermore, the alignment of the camera and the direction of motion also affects the feature matching results since changes in feature scales will not be the same in different configurations.

It is known that the result of triangulation is more accurate for objects that are close to the camera [39], as the camera requires little motion to observe these objects from different viewpoints. For objects that are further ahead, the accuracy of depth estimation degrades as the distance of the camera to the objects increases. This situation is illustrated in Figure 4.14 where two configurations are shown in a scene that includes three objects at different distances from the camera.



(a) Camera direction is perpendicular to the direction of motion.

(b) Directions of camera and motion are parallel to each other.

Figure 4.14: Alignment of the camera relative to the direction of motion. Of the three objects in the scene, the house is the closest object and the star is the furthest with the tree located between them. Second frames in both figures show the previous projected positions of the objects to demonstrate the difference. Notice how the disparities of these objects are changing in their image projections as the camera moves in different directions.

In the first configuration (Figure 4.14(a)), the direction of the camera is perpendicular to the motion direction. In this case, features from the nearby object (house) have a larger disparity between the frames. Disparities from the tree are less significant, and there is no visible disparity in the feature from the star.

The challenge appears when the camera is pointing in the direction of motion as in Figure 4.14(b). In this configuration, the baseline shortens and it becomes more difficult to observe parallax, even for nearby objects, and this becomes almost impossible for objects residing further ahead.

A second challenge of using such a configuration is related to changes in feature scales. In a horizontal motion (*i.e.* when the camera is not parallel to the direction of motion), the scales of features will not change much; rather, the changes will be in position. However, in a vertical type of motion as shown in Figure 4.14(b), feature scales will vary greatly. This situation poses a challenge for the feature descriptors, particularly for their scale-invariance properties mentioned in section 2.1.2. For this reason, descriptors that are used in the vision system should be able to accommodate such great changes in scale.

Despite the two challenges explained above, the second configuration was used in the experiments, because the images acquired by the camera are not only used for user tracking but also used as the background scene for the AR applications presented in chapter 7. A setting with the former configuration would not allow this using only a single camera.

4.4.2 Extraction of keyframes

When performing localization on a video sequence, it is useful to select a subset of the frames, a process known as keyframing. In this process, it is important

to select these keyframes so that the number of feature correspondences between them is above some threshold, so that the system can estimate the motion correctly and does not skip any important movement. Keyframing offer the following two benefits:

- Large baseline: The estimation of 3D structure is not accurate when the baseline is small (Figure 4.14) *i.e.* there is not enough motion between the keyframes.

- Performance: The complete algorithm is run on a smaller set of images (using 40–60% of the frames in this case) instead of the complete sequence.

The keyframing approach described in the following paragraph is based on [85, 253]. In addition to their method, an extra heuristic was used to prevent skipping too many frames as this may result in missing any sudden motion that may have lasted for just a few frames.

The first image acquired by the camera, $frame_0$, is always selected as the first keyframe ($keyframe_0$). For the following frames, a set of feature correspondences were computed between the last keyframe $keyframe_n$ and each of following frames $frame_i$. Spurious matches are eliminated from this set using RANSAC. When the number of these correspondences falls below a threshold ($t = 400$, found experimentally), a new keyframe is extracted as $keyframe_{n+1}$. This selection mechanism is illustrated in Figure 4.15.

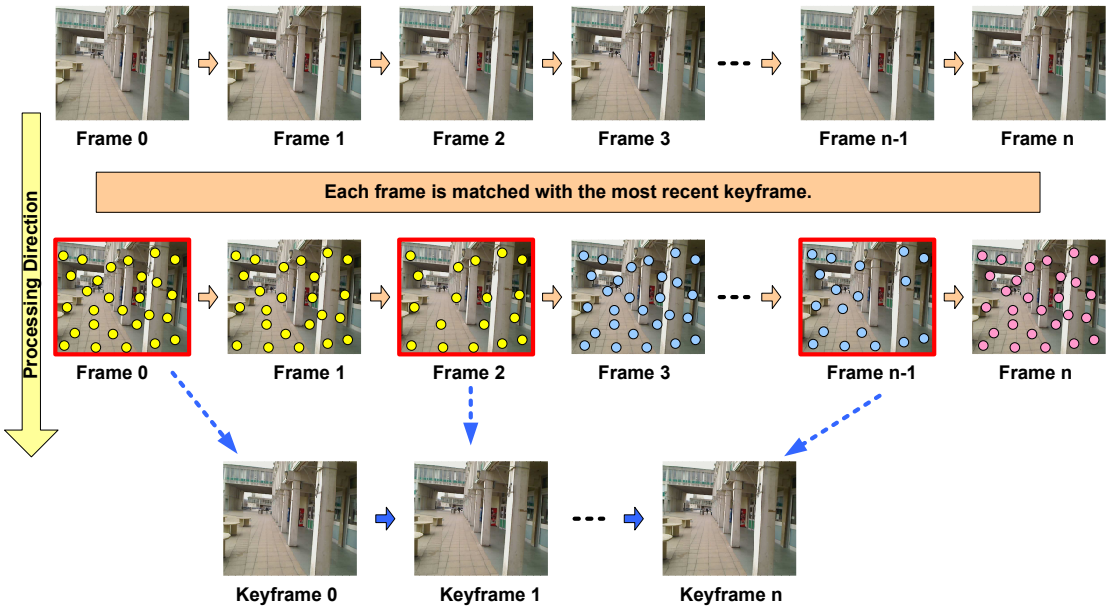


Figure 4.15: Selecting keyframes based on image correspondences. Coloured circles represent feature correspondences between frames acquired by the camera and the most recent keyframe. When the number of these correspondences fall below a threshold, a new keyframe is extracted and used for motion estimation.

After each keyframe is extracted, the number of skipped frames is counted, allowing at most 3 frames to be skipped.

This method of extracting keyframes uses 40% – 60% of the frames, allowing better speed performance and providing a large enough baseline for the motion estimation. The next steps include finding possible solutions for the transformation between the most recent two keyframes and then selecting the most appropriate one.

4.4.3 Finding possible solutions

The localization of the camera is performed by using the essential matrix instead of using the fundamental matrix, making use of the calibration parameters [53] found in section 4.1 and follows the two-view geometry approach described in [31]. This method finds the camera motion parameters camera motion parameters R and t as shown in Figure 4.16.

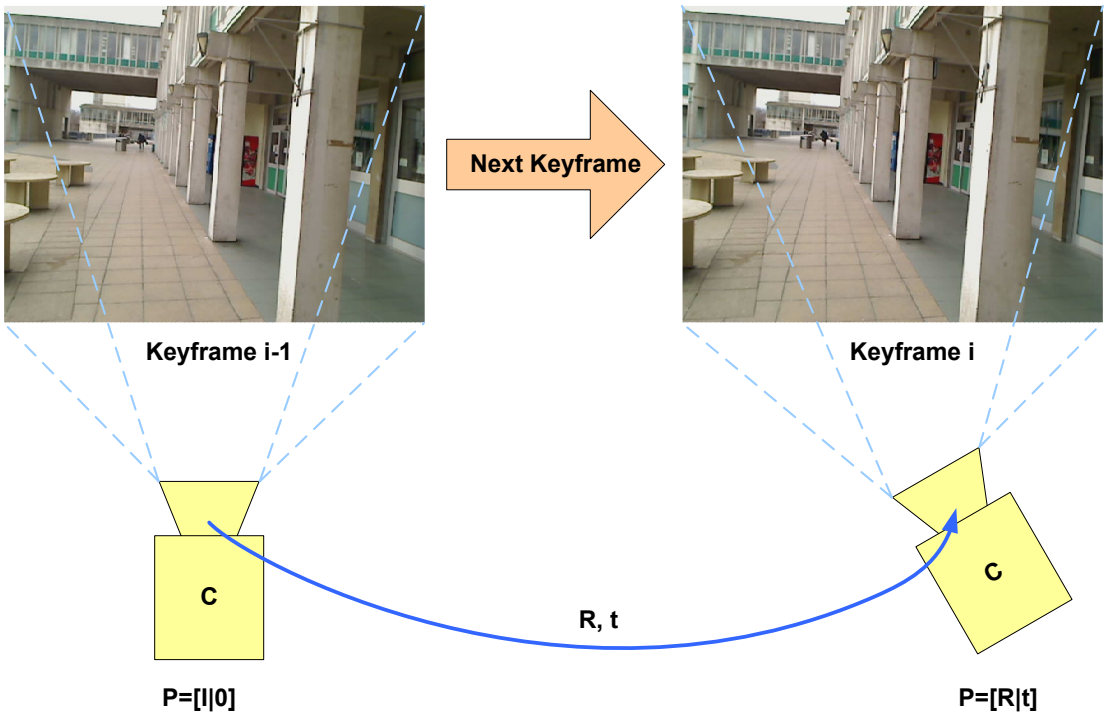


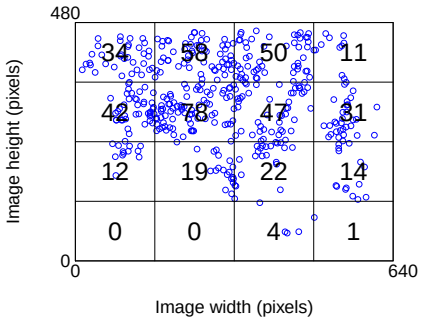
Figure 4.16: Motion parameters for camera

When a keyframe $keyframe_i$ is obtained, it is then used to calculate the motion of the camera since the previous keyframe $keyframe_{i-1}$. For this purpose, first the feature point correspondences between these two keyframes are obtained from the correspondences that were already calculated to extract the most recent keyframe using the algorithm of section 4.4.2.

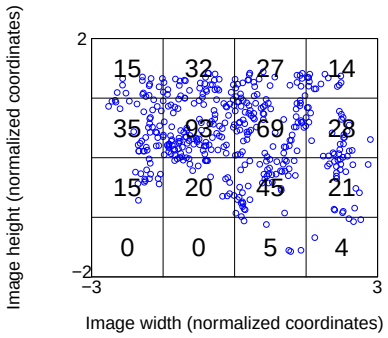
These correspondences have a normalizing transformation [258] applied to them, which is obtained by first calculating the translation which moves the points around origin and then applying a scaling transformation so that the mean distance of points to the origin $(0, 0)$ will be $\sqrt{2}$. This transformation is given as

$$T_{norm} = \begin{bmatrix} s_t & 0 & -s_t\bar{x} \\ 0 & s_t & -s_t\bar{y} \\ 0 & 0 & 1 \end{bmatrix} \tag{4.22}$$

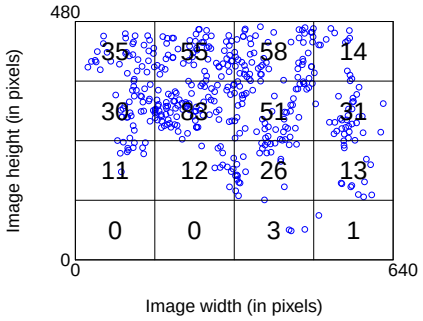
where s_t is the scaling parameter which is calculated using the distances of features to the origin while $\bar{x}$ and $\bar{y}$ are the coordinate averages. This transformation is applied to the features in both $keyframe_i$ and $keyframe_{i-1}$. This normalizing transformation is an important step since it affects the conditioning of the matrices used in calculating the fundamental matrix [31]. The transformation is calculated separately for both of the keyframe features (shown earlier in Figure 3.2) and it can be seen that a more uniform distribution of the features is obtained as a result of the transformation as shown in Figure 4.17.



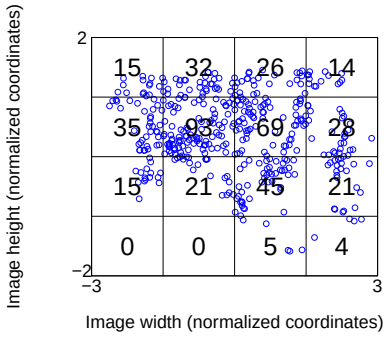
(a) Feature locations from Image 1



(b) Normalized coordinates for Image 1



(c) Feature locations from Image 2



(d) Normalized coordinates for Image 2

Figure 4.17: Effect of normalizing feature positions that were extracted in Figure 3.2. It can be seen that features are distributed more uniformly after normalization.

It is also interesting to see how the results of a normalizing transformation are in line with the evaluation presented in chapter 3 of this thesis.

Following this normalization, the fundamental matrix F, which defines the two-view geometry, was calculated using these normalized coordinates. The calculated fundamental matrix was denormalized using

$$F_{denorm} = T_i F T_{i-1} \tag{4.23}$$

where T_{i-1} and T_i are the normalizing transformations for keyframes $i-1$ and i respectively.

Before calculating the essential matrix [252] E, it is important to make sure that F satisfies the rank-2 constraint (F having zero determinant) since this motion estimate is sensitive to noise from small motion and feature points that yield a degenerate surface *e.g.* a plane in 3D [259]. This is done (following the approach described in [259]) by first calculating the SVD of F as $F = USV^T$. The next step is setting the last element of S matrix (*i.e.* $S_{3,3}$) to 0. Finally, F is recreated using the multiplication of this modified S: USV^T.

When the camera calibration K (described in section 4.1) is known, the essential matrix E can be calculated from the fundamental matrix [31] using

$$E = K^T F K \tag{4.24}$$

Using E, the motion parameters can be obtained up to scale and with some uncertainty using the approach described in [31]. This will result in more robust computations [53] due to known camera calibration. The uncertainty arises from the existence of four possible solutions for R and t. These solutions indicate four

possibilities in which the position of features differ relative to the camera position.

For this purpose, the following two matrices known as Hartley matrices [31] are defined as

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad Z = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4.25}$$

where W is an orthogonal matrix ($W^T W = I$) and Z is a skew-symmetric matrix ($Z^T = -Z$). These two matrices will be used to calculate solutions for the camera motion parameters. For this purpose, first the SVD of E is calculated

$$E = USV^T \tag{4.26}$$

Then, the following intermediate matrices are calculated using the matrices of (4.25) with U and V orthogonal matrices obtained from (4.26)

$$T = UZU^T \quad R_1 = UWV^T \quad R_2 = UW^TV^T \tag{4.27}$$

For R_1 and R_2, the determinant is made positive [260]

$$\begin{aligned} &\text{if } |R_1| < 0 \text{ then } R_1 = -R_1 \\ &\text{if } |R_2| < 0 \text{ then } R_2 = -R_2 \end{aligned} \tag{4.28}$$

Then, a vector t' is created using elements of the T matrix given in (4.27)

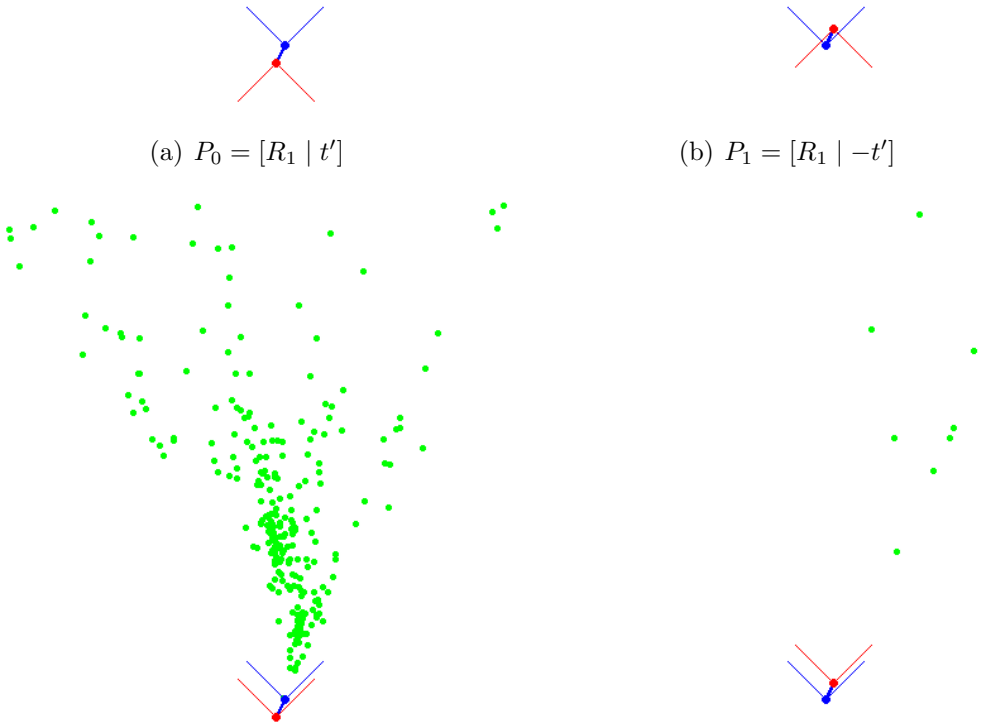
$$t' = \begin{bmatrix} T_{2,1} \\ T_{0,2} \\ T_{1,0} \end{bmatrix} \tag{4.29}$$

Now, the four solutions can be obtained as

$$\begin{aligned} P_0 &= [R_1 \mid t'] \\ P_1 &= [R_1 \mid -t'] \\ P_2 &= [R_2 \mid t'] \\ P_3 &= [R_2 \mid -t'] \end{aligned} \tag{4.30}$$

One of these four solutions will be selected as the motion estimate between the most recent two keyframes; the most appropriate one is found by creating a 3D reconstruction of the feature points and then projecting them onto the images using the projection matrices constructed with these motion parameters (R and t).

Figure 4.18 shows a visualization of the four solutions obtained from (4.30) along with the triangulated points calculated as described in section 4.4.4.



(a) $P_0 = [R_1 \mid t']$ (b) $P_1 = [R_1 \mid -t']$

(c) $P_2 = [R_2 \mid t']$ (d) $P_3 = [R_2 \mid -t']$

Figure 4.18: Four solutions and triangulation results given in (4.30) following [31]. Blue circle indicates the initial camera position given with $P_I = [I \mid 0]$ and red circle shows the new position and orientation of the camera ($P_T = [R \mid t]$). Note that (a) and (b) does not have any triangulated points due to the new orientation of the camera while (c) and (d) shows triangulated points. The solution of (c) will be selected as the motion estimate since it has the largest number of 3D inliers.

4.4.4 Triangulation of features

The feature points are reconstructed in 3D by triangulation using the method presented in [261]. The method triangulates a feature point using two camera matrices, P_I and P_T, where the former defines the origin for the motion (for $keyframe_{i-1}$) whereas the latter uses the motion parameters of (4.30) for $keyframe_i$:

$$P_I = K\,[I \mid 0] \tag{4.31}$$

and

$$P_T = K\,[R \mid t] \tag{4.32}$$

where K is the calibration matrix created using the parameters from the process explained in section 4.1.

The initial camera matrix is assumed to be $P_I = [I \mid 0]$ and the motion calculated using the solutions above is relative to this initial point. The next thing to do is to decide on the motion defined by P_T as illustrated in Figure 4.18.

An image point is triangulated, using the linear triangulation method of [31], as follows: First the two projection matrices P_I and P_T are created so that the camera calibration is incorporated in the projection matrices. Then each feature is triangulated by creating the following matrix for $j \in (0, 1, 2, 3)$ following the approach of [254]:

$$J = \begin{bmatrix} P_I[2,j]f_{I1_x} - P_I[0,j] \\ P_I[2,j]f_{I1_y} - P_I[1,j] \\ P_T[2,j]f_{I2_x} - P_T[0,j] \\ P_T[2,j]f_{I2_y} - P_T[1,j] \end{bmatrix} \tag{4.33}$$

In (4.33), f_{I1_x}, f_{I1_y} and f_{I2_x}, f_{I2_y} correspond to the image positions for a feature

f in images 1 and 2 respectively. $P_I[i,j]$ and $P_T[i,j]$ correspond to the element in cell $[i,j]$ of the related projection matrix.

The SVD of J matrix is calculated

$$J = USV^T \tag{4.34}$$

and 3D position of the triangulated feature (represented by X) is obtained from the last column of V:

$$X = \begin{bmatrix} V[0,3] \\ V[1,3] \\ V[2,3] \\ V[3,3] \end{bmatrix} \tag{4.35}$$

Once the image points are triangulated between the two views, these 3D points can be used to select one of the solutions produced above. This is done by projecting the 3D points and checking whether they are in front of the camera in both views (P_I and P_T) or not as follows. The projection of X is calculated in both views:

$$\begin{aligned} x_{P_I} &= P_I X \\ x_{P_T} &= P_T X \end{aligned} \tag{4.36}$$

This projection is then checked to determine whether the z coordinate is positive relative to the cameras:

$$\begin{aligned} x_{P_I}[2,0]\ X[3,0] > 0 \\ x_{P_T}[2,0]\ X[3,0] > 0 \end{aligned} \tag{4.37}$$

The logic in (4.37) finds 3D inliers by checking whether a feature has a positive depth relative to the camera (*i.e.* in front of the camera). The solution resulting in the largest number of inliers is selected as R and t for the final motion estimate.

From the solutions given in Figure 4.18, it can be seen that P_2 has the largest number of 3D inliers so this solution is selected as the estimated transformation. Projections of the triangulated points have an mean error of 0.20 pixels on both images.

4.4.5 Final motion estimate

The translation parameters can directly be obtained from t from the selected solution but in order to extract rotations about x, y and z axes from the rotation matrix R, Rodrigues' formula [262] was used. According to this, $R[ij]$ is the corresponding index of the rotation matrix and the rotations are calculated using

$$\begin{aligned} R_y &= \sin^{-1}(R[0,2]) \\ R_x &= \sin^{-1}(-R[1,2])/\cos R_y \\ R_z &= \sin^{-1}(-R[0,1])/\cos R_y \end{aligned} \tag{4.38}$$

From these found transformations, a transformation matrix is formed as

$$Tr = \begin{bmatrix} \cos R_y \cos R_z & -\cos R_y \sin R_z & \sin R_y & t_x \\ \sin R_x \sin R_y \cos R_z + \cos R_x \sin R_z & -\sin R_x \sin R_y \sin R_z + \cos R_x \cos R_z & -\sin R_x \cos R_y & t_y \\ -\cos R_x \sin R_y \cos R_z + \sin R_x \sin R_z & \cos R_x \sin R_y \sin R_z + \sin R_x \cos R_z & \cos R_x \cos R_y & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.39}$$

This transformation is then applied to the last position of the camera stored in homogeneous coordinates. This approach is similar to dead-reckoning [255], where the final position is calculating by adding the estimated displacement to the last known position.

As mentioned earlier, a filtering process using KF was used before this transformation matrix was created in order to refine the obtained estimate. This part

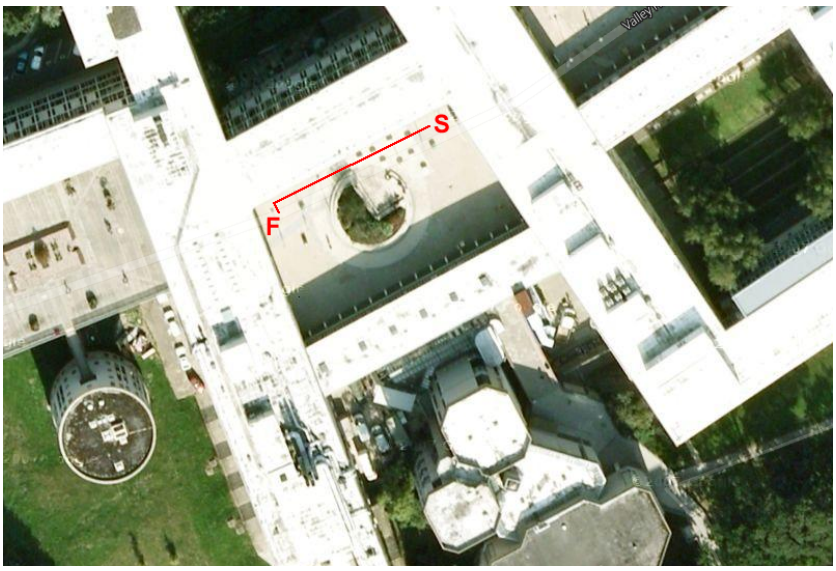
was removed from the implementation since this result will be combined with results from other sensors in chapter 6. The results are presented in the following section.

4.5 Results

This section presents results for the developed keyframe-based motion estimation method. Figure 4.19 presents the trajectory results for the first 500 frames of a video sequence in which the user, wearing a camera-attached helmet, walks initially in a straight line and then, turns left near the end of the trajectory. Two descriptors (SURF and ORB, mentioned in section 2.1.2) are used for feature detection and description mainly due to their real-time performance. (ORB is a little faster than SURF.)

From the results, it can be seen that SURF resulted in a better estimation of the motion than ORB since the latter could not estimate the motion in z direction (the direction in which the camera is pointing). It is suspected that this is because the scale invariance of SURF is superior to that of ORB.

Figure 4.20 shows the estimated trajectory of the user on a curved path. It can be seen that ORB is performing relatively better than for the previous dataset since the motion has a horizontal component as well as a vertical one. A jump in the position can be seen in Figure 4.20(c): ORB is failing to estimate the motion correctly.



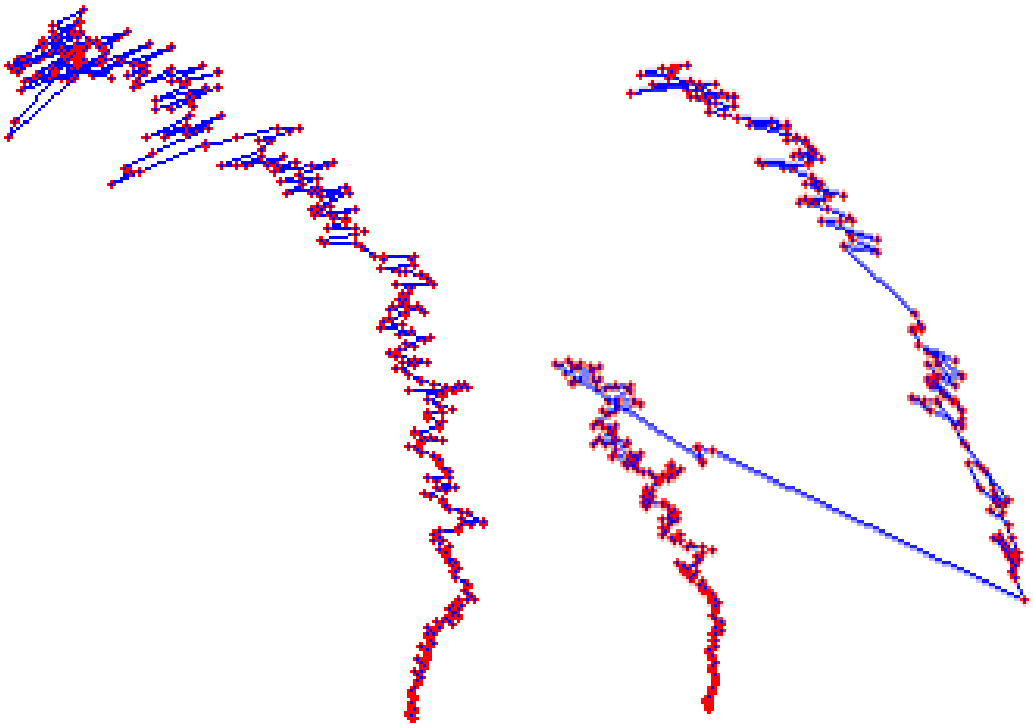
(a) Ground truth for the straight path

(b) SURF path

(c) ORB path

Figure 4.19: Camera trajectories using SURF and ORB for a camera moving on a straight path and turning to left at the end. ORB fails to estimate the motion in z direction.

(a) Ground truth for the curved path



(b) SURF path

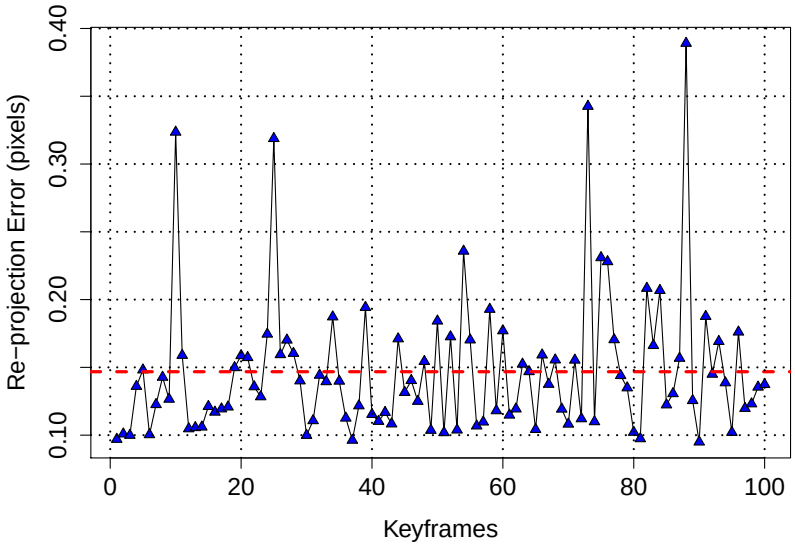
(c) ORB path

Figure 4.20: Camera trajectories using SURF and ORB for a camera moving on a curved path. Note the increase in error in later frames due to error accumulation shown by longer blue lines.

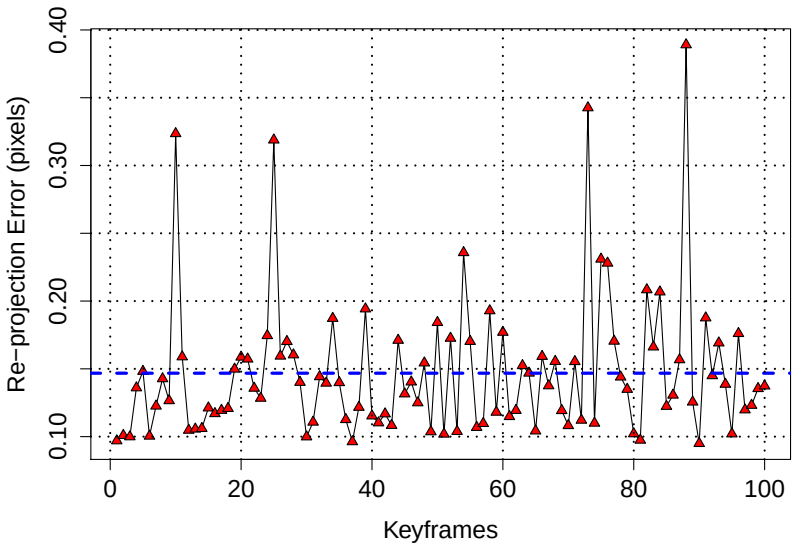
A second important thing to mention in the results for the curved sequence (Figure 4.20) is the increasing distance between estimated positions, even though the speed of the user was constant during the experiment. This is due to the accumulation of errors in the dead-reckoning approach used and is clearly visible in later frames of the sequence.

By looking at the re-projection errors for the first 100 keyframes extracted from the straight path sequence (shown in Figures 4.21 and 4.22), it can be seen that SURF performed better than ORB. SURF has a mean error of less than 0.15 pixels for the sequence, whereas this error for ORB exceeded 1 pixel for some of the frames; its mean error is about 0.5 pixels.

For the curved path sequence, SURF results in a larger projection error. This is less than 2 pixels on average but can go up to 12 pixels (Figure 4.23). ORB results exhibit a lower projection error (less than one on average, as shown in Figure 4.24) until the large position jump (Figure 4.20(c)). After this point the error for ORB goes up to 335.84 pixels.

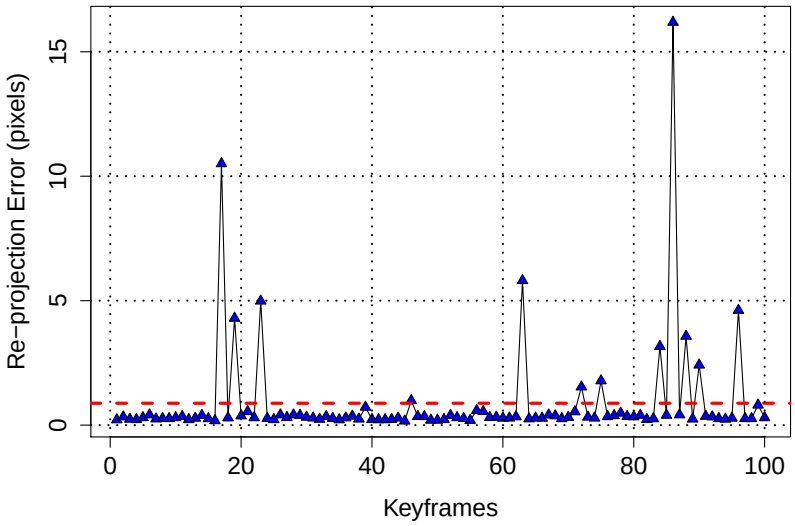


(a) Projection using $P_{init} = [I \mid 0]$

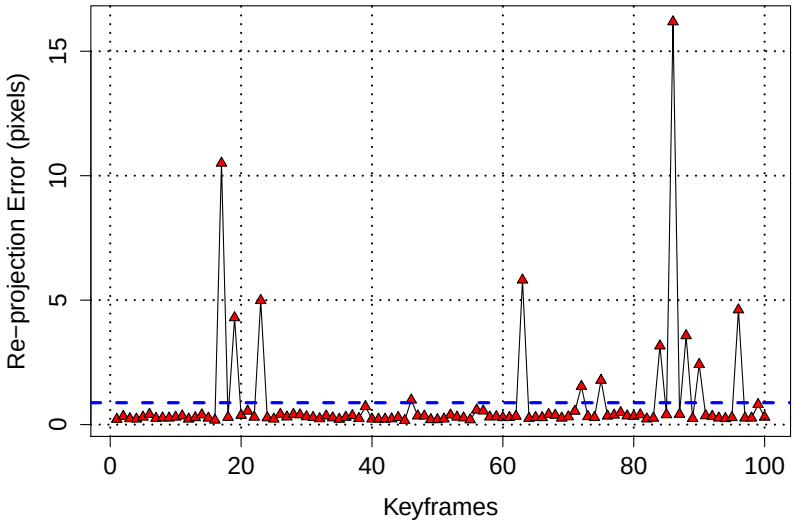


(b) Projection using $P_{new} = [R \mid t]$

Figure 4.21: Re-projection errors for SURF for the straight dataset through the first 100 keyframes

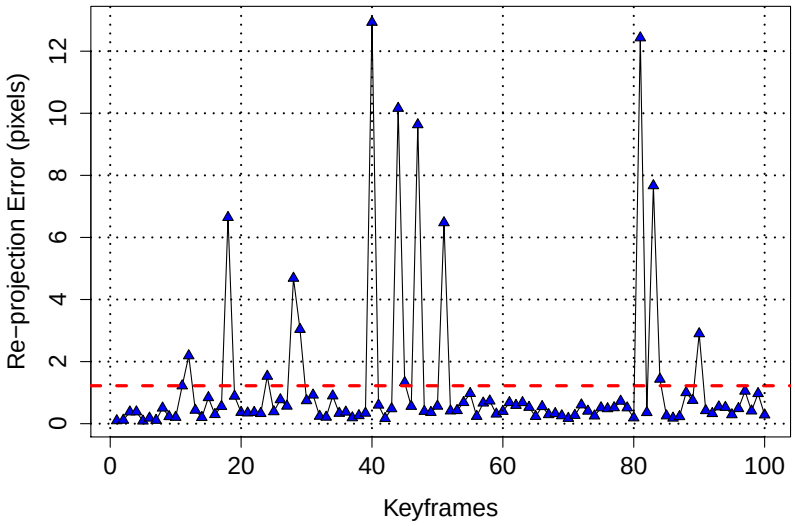


(a) Projection using $P_{init} = [I \mid 0]$

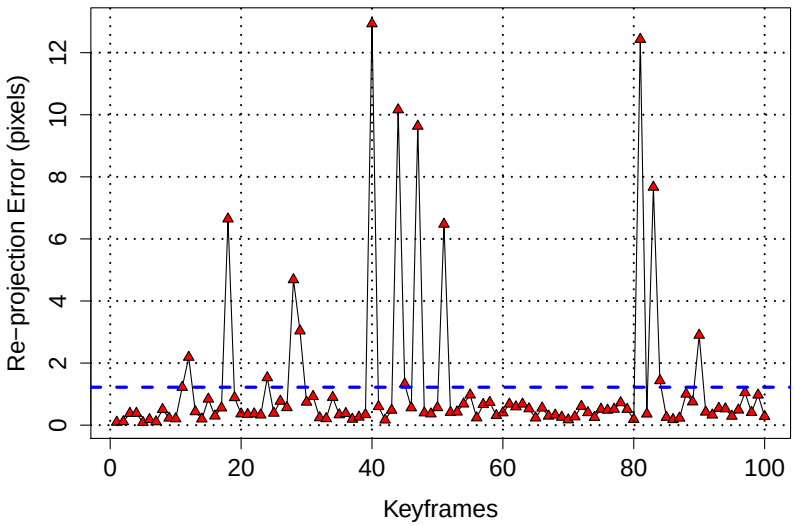


(b) Projection using $P_{new} = [R \mid t]$

Figure 4.22: Re-projection errors for ORB for the straight dataset through the first 100 keyframes

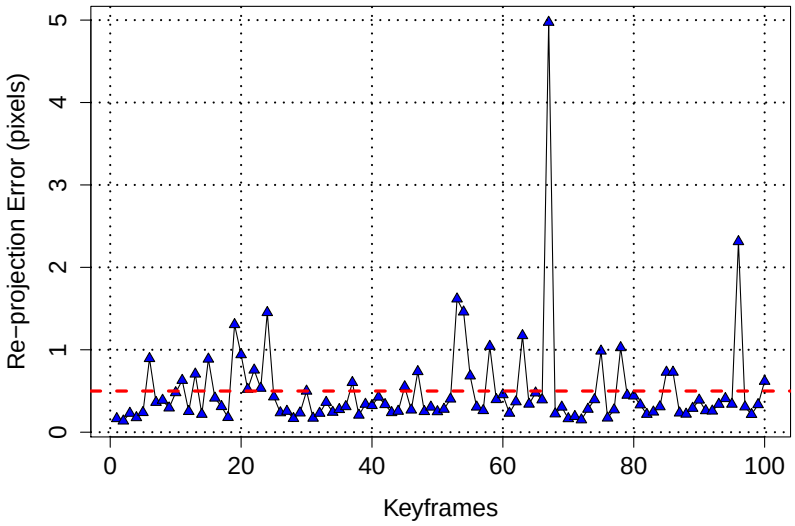


(a) Projection using $P_{init} = [I \mid 0]$

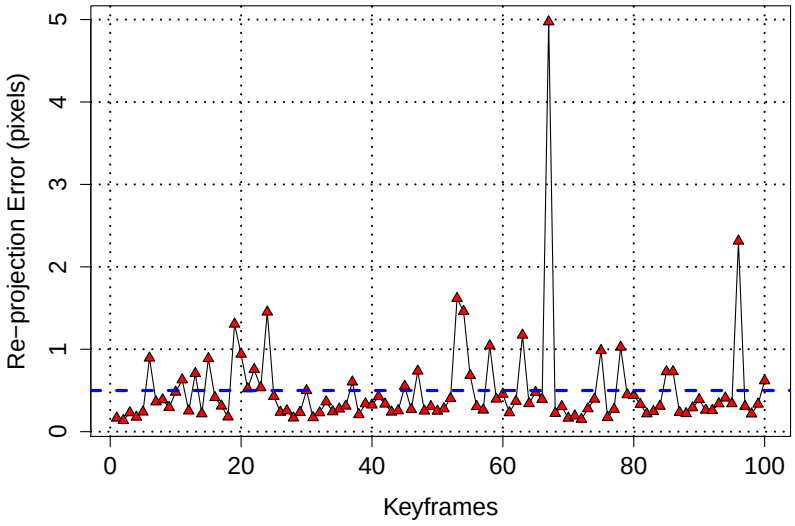


(b) Projection using $P_{new} = [R \mid t]$

Figure 4.23: Re-projection errors for SURF for the curved dataset through the first 100 keyframes



(a) Projection using $P_{init} = [I \mid 0]$



(b) Projection using $P_{new} = [R \mid t]$

Figure 4.24: Re-projection errors for ORB for the curved dataset through the first 100 keyframes

4.6 Remarks

This chapter examined vision-based methods for estimating the motion of a walking user. The discussion started with camera calibration, finding the intrinsic parameters of a camera using a sequence of images of a calibration grid taken when moving the grid in front of the camera. The parameters found by this calibration process are used to model the camera.

The discussion continued with filtering methods namely, KF and PF, which are used for state estimation problems. An example application of tracking a virtual user walking along a sinusoidal path was provided in order to visualize their operation and eventually provide a comparison of them.

A SLAM implementation using EKF was analysed for tracking a user in an outdoor environment. Details of the processing steps were given and experimental results were presented. This method was accurate in tracking the motion as long as feature matches were correct and the motion was slow. However, when the camera performs erratic motions, such as severe rotations, the method fails due to data association problems. More problems were faced used when this implementation was used in an outdoor environment since the number of features extracted increased significantly during operation, slowing down the filter performance and reducing tracking accuracy.

An alternative approach that makes use of two-view geometry was presented for estimating the motion of the user. The discussion started by examining the assumptions behind this approach and the challenges related to the position of the camera in relation to the direction of motion. The proposed approach used keyframes extracted by a simple algorithm and estimated the motion between the most recent keyframes. This motion estimate is then applied in the form of a

transformation to the most recent position of the user, as in dead-reckoning.

The use of two different feature descriptors (SURF and ORB) were compared in terms of tracking accuracy. It was observed that SURF was better in motion estimation with two datasets captured outdoors. ORB missed some of the forward movements but performed better in the case where the motion also had a horizontal component. Overall, the estimate obtained by the SURF descriptor was found to be closer to the actual motion. The trade-off introduced with the SURF descriptor is the additional computational expense.

It was also found that the accumulation of errors caused inaccuracies in tracking in later frames, a consequence of the dead-reckoning approach followed.

The problems related to speed performance and error accumulation are tackled in chapter 6 by employing additional sensors in a multi-threaded approach in order to provide a more accurate motion estimate.

The next chapter examines the use of the Kinect sensor for user tracking and object detection in indoor environments, again using vision-based approaches. These are somewhat different from those described in this chapter since they are intended for indoor environments where the large baseline requirement for the localization algorithm would not be easily satisfied.

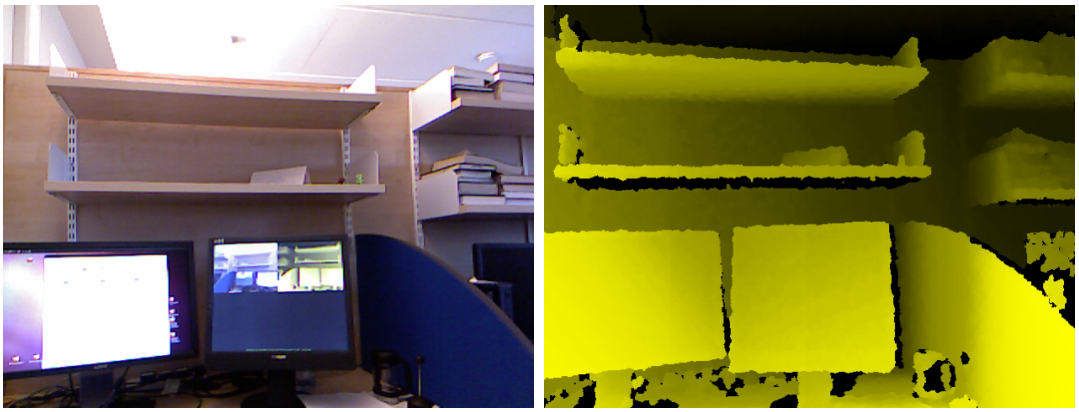
CHAPTER 5

VISION WITH DEPTH SENSOR

The vision-based motion estimation approach presented in the previous chapter requires a large baseline in order to estimate the depth of feature points accurately and hence identify the motion of a user. Indoor environments are not usually suitable for obtaining such a large baseline, especially for cases when the user is mostly standing or making small movements.

Microsoft's RGB–depth sensor Kinect [263] provides depth (range using IR light) information along with the RGB image which can be used to generate a 3D point cloud as a representation of the environment as shown in Figure 5.1.

Although point clouds are useful in presenting the 3D structure of the surrounding, handling large numbers of points is not efficient in terms of time and space complexity. Often, a higher level geometry, *e.g.* a plane, is a better description of the environment. For this reason, planar structures from point clouds have been used in different application types such as robot navigation [196],



(a) RGB image (b) Depth image

Figure 5.1: RGB and depth images from Kinect. In (b), objects closer to the sensor are shown with a brighter yellow while the colour becomes darker for the objects away from the sensor.

AR [148, 264] or hybrid rendering [265].

A second advantage offered by using planes is that they can be stored and handled easily, using only four parameters per plane rather than storing hundreds of thousands of point features. RANSAC [71] is well known to be successful at finding planes from 2D or 3D point sets/clouds. A RANSAC based algorithm for finding 3D primitive objects from large point clouds is presented in [266]. A modification of this algorithm is used to extract planar features obtained from a laser scanner in [267]. A second method for finding planes is the 3D Hough Transform which has been recently applied for depth data from Kinect in [268] where a random point was used with two other points selected from the first point's circular boundary.

With Kinect it is also possible to extract the pose of the camera using the PnP methods described in section 2.1.3. This pose can be used as the viewpoint for the virtual camera which will be used for rendering synthetic objects over real-world objects for an *in situ* AR algorithm.

Estimation of the camera pose in 3D is a widely-studied research topic [25, 31, 78] (see section 2.1.3), normally obtained using vision-only methods such as the PnP solution, which aims to recover camera pose using the positions of 3D features and their projections. For instance, Tam *et al.* [269] used OpenCV's PnP functionality for real-time AR: In a preliminary (off-line) phase, a map of features was first created from known 3D positions in the real world, with each point being described by a SURF descriptor [58]. Subsequently, an on-line system matched SURF features from the camera to those of the map, and then the camera pose was calculated. The time taken for calculating and matching SURF features is significant, so this had to be performed on a Graphical Processing Unit (GPU).

Sensing the colour and depth of a scene concurrently with Kinect allows extracting the 3D structure of the surrounding and creating a representation of the environment. The use of Kinect data with the PnP algorithm has also been investigated [270] for image-based registration. The analysis involved determining relative and absolute accuracies of the Kinect and another sensor according to the camera pose obtained from the E-PnP algorithm [89].

Using these features of Kinect, the chapter investigates two approaches of detecting planar objects in the scene and finding camera pose again using scene features. The discussion starts by explaining the importance of calibrating the Kinect in order to align the RGB and depth images from its sensors in section 5.1.

Then, section 5.2 describes a simple sequential algorithm which extracts planar features only from the depth data by trying to define a plane using a random selection of 3D points, defining a plane and finding the maximal number of points that are lying on this plane as a support for the extracted plane. Later, an approach using several threads in order to find planes concurrently will be described for modifying the initial algorithm in order to utilise the hardware resources in a

better way and reducing the runtime.

The discussion is continued in section 5.3 by describing a method for storing 3D–2D point correspondences efficiently using a simple data structure. These correspondences are also used with a recent PnP solution in order to find the pose of the camera. A KF is then used to refine the initial position estimate of the camera since the noise from the IR sensor manifests itself as "jittering" in the output. Having found the camera pose that will be used as the viewpoint for an AR application (presented in chapter 7), the next step is finding a method for extracting rectangular structures satisfying some conditions (*e.g.* aspect ratio, *etc.*). These detected rectangles are tracked using a PF in order to provide stability in the application mentioned. Then, the results for this approach will be presented.

The last experimental work presented in this chapter is on human skeleton tracking features of Kinect in section 5.4, which will be used to track certain body joints for another interesting AR application presented later in the thesis.

Finally, the chapter will be concluded in section 5.5.

5.1 Sensor Calibration

Conventional camera calibration, described in section 4.1, focuses on the calibration of a single camera in order to find its intrinsic parameters (*e.g.* focal length, optical centre, *etc.*). Some of these parameters (*e.g.* distortions) model the impurities/inaccuracies in the manufacturing process of the camera lens. Similar problems can also be seen in the IR sensor of Kinect (see Figure 2.2). In addition to finding the internal parameters for Kinect's both RGB and depth sensors, the distance between these two is an additional parameter to consider. All these parameters result in different projected locations in the RGB and the depth images

for the same 3D point, as shown in Figure 5.2.

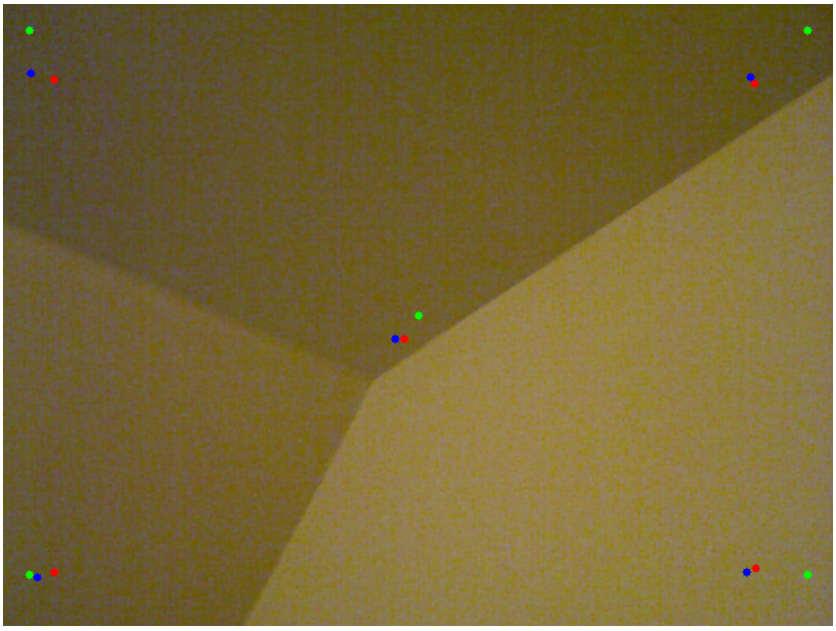


Figure 5.2: Projection differences introduced by the calibration parameters. Green points show the selected pixel position from the depth image. Red and blue points show the projections when the radial and tangential distortions are ignored and considered respectively.

The projection of depth data into world coordinates and re-projection from world coordinates onto the RGB image requires the calibration parameters of both sensors in advance if errors are to be minimized [271]. These calibration parameters can be computed either using the Kinect Calibration Toolbox [37] or obtained using a method [38] similar to stereo calibration [39]. The first method provides a semi-automatic way of calibrating the device, by allowing the user to select corners manually from a calibration target (*e.g.* chess-board pattern in Figure 4.2) in the RGB image and the plane where the calibration target is placed in the depth image. These selections are used as an initial guess for the calibration parameters for a set of RGB and depth images ($\sim$ 30 images). A

non-linear minimization method is then used to reduce the projection errors.

This work uses the calibration parameters found by the second method [38]. It again finds corners on the calibration target and performs the calibration, but only for the RGB camera. The depth camera is then calibrated by manually finding the corners in the depth image. The values for these calibration parameters and transformation matrices are given in Table 5.1.

Table 5.1: Calibration parameters for Kinect

		x	**y**	
Focal length	**RGB**	529.22	525.56	
	Depth	594.21	591.04	
Principal point	**RGB**	328.94	267.48	
	Depth	393.31	242.74	
		k1	**k2**	**k3**
Radial coefficients	**RGB**	0.26452	-0.83990	0.91192
	Depth	0.26386	0.99967	-1.30540
		p1	**p2**	
Tangential coefficients	**RGB**	-1.9922 $\times 10^{-3}$	1.4372×10^{-3}	
	Depth	-7.6276×10^{-4}	5.0350×10^{-4}	

The calibration parameters are used to construct the camera matrices $\mathbf{K_D}$ and $\mathbf{K_C}$ for the depth and RGB sensors respectively as well as the distortion coefficients vector $\mathbf{dc_C}$:

$$\mathbf{K_D} = \begin{bmatrix} f_{x_D} & 0 & c_{x_D} \\ 0 & f_{y_D} & c_{y_D} \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{K_C} = \begin{bmatrix} f_{x_C} & 0 & c_{x_C} \\ 0 & f_{y_C} & c_{y_C} \\ 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

$$\mathbf{dc_C} = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} \tag{5.2}$$

where f_x and f_y are the focal lengths in the x and y directions and c_x and c_y are the centre coordinates of the relevant sensor. k_1, k_2, k_3 are the coefficients for radial distortion and p_1, p_2 are the tangential distortion parameters.

Transformations ($\mathbf{R}$ and $\mathbf{T}$ in (5.3) and (5.4)) between the depth and RGB sensors of the Kinect are used to align the outputs of two sensors and are obtained as calibration results.

$$\mathbf{R} = \begin{bmatrix} 9.9985 \times 10^{-1} & 1.2635 \times 10^{-3} & -1.7487 \times 10^{-2} \\ -1.4779 \times 10^{-3} & 9.9992 \times 10^{-1} & -1.2251 \times 10^{-2} \\ 1.7470 \times 10^{-2} & 1.2275 \times 10^{-2} & 9.9977 \times 10^{-1} \end{bmatrix} \tag{5.3}$$

$$\mathbf{T} = \begin{bmatrix} 1.9985 \times 10^{-2} \\ -7.4424 \times 10^{-4} \\ -1.0917 \times 10^{-2} \end{bmatrix} \tag{5.4}$$

The intrinsic parameters, distortion coefficients and the transformations will be used to obtain 3D points from raw depth values in the depth image from Kinect for finding planar objects in the following sections.

5.2 Plane Extraction Using Depth Sensor

It was mentioned earlier in the chapter, that a higher level structure such as a plane is usually a better description of the environment than a point cloud [10]. Initially, finding and extracting 3D planar features using the depth sensor seemed promising for the *in situ* augmentation application described in chapter 7.

Instead of using the 3D Hough Transform [268], the approach presented here uses the explicit definition of a 3D plane [272] obtained from a triplet of points,

$\mathbf{p_0}$, $\mathbf{p_1}$ and $\mathbf{p_2}$) lying on a plane, as shown in Figure 5.3.

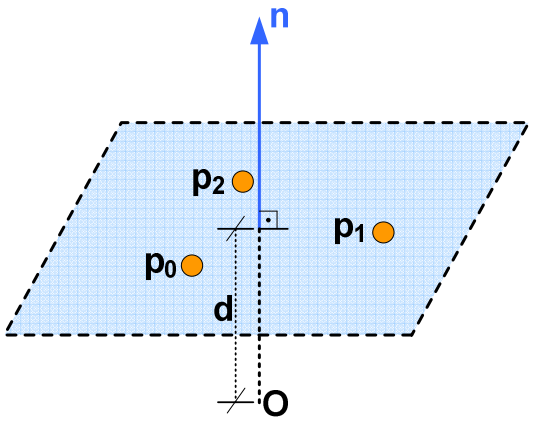


Figure 5.3: Parameters for the explicit definition of a plane. The definition uses the normal vector $\mathbf{n}$ and the distance parameter d. Three main points ($\mathbf{p_0}$, $\mathbf{p_1}$ and $\mathbf{p_2}$) are used to define these parameters.

This definition uses one point on the plane ($\mathbf{p_0}$), the normal vector ($\mathbf{n}$) to the plane which indicates plane's direction and the distance parameter (d) that lies along the normal vector and denotes the minimum distance of the plane to the origin (O). The normal vector is calculated as

$$\mathbf{n} = (\mathbf{p_1} - \mathbf{p_0}) \times (\mathbf{p_2} - \mathbf{p_0}) \tag{5.5}$$

A plane can then be defined using any of the points from the triplet, for example

$$\mathbf{p_0}.\mathbf{n} + d = 0 \tag{5.6}$$

The main assumption of this procedure is that the points are neither coincident nor collinear which must be checked for the selected triplets as will be described later.

Given a depth image of 640×480 pixels and the corresponding RGB image

of the same size (as in Figure 5.1), the plane extraction algorithm first finds the world coordinates from the depth data and creates a list of 3D points. This list is then given to the plane extraction algorithm in which each valid triplet is examined to determine whether they can define a plane. (The validity of a triplet will be discussed later in the chapter.) Once a plane is found, it is added to a list of planes which will be the output of the algorithm. The process is outlined in Algorithm 2.

Algorithm 2 Plane extraction

Require: The list of 3D points in world coordinates P
  $\Pi \leftarrow \emptyset$, the list of extracted planes
  $\langle \mathbf{p_0}, \mathbf{p_1}, \mathbf{p_2} \rangle \leftarrow rand(P)$, find a triplet from P
  $C_\pi \leftarrow \langle \mathbf{p_0}, \mathbf{p_1}, \mathbf{p_2} \rangle$, generate a candidate plane from the triplet
  if $C_\pi \sim \pi_i$ in Π **then**
    Discard C_π, a similar plane already exists
  else
    if $|C_\pi| \geq \tau$ **then**
      $\Pi \leftarrow C_\pi \bigcup \Pi$, add the candidate plane to the list
    else
      Discard C_π, could not satisfy min # of points
    end if
  end if
  return Π

After extracting planes, they can be projected on the RGB image along with the list of points used to extract them. Details of the processing steps are presented in the following subsections.

5.2.1 Finding world coordinates from a depth image

Kinect returns an RGB image along with a depth image which are not perfectly aligned due to the disparity between the sensors as mentioned in section 5.1. In order to find correspondences, first the values from the depth image must be back-

projected into 3D coordinates and then projected onto the RGB image. Figure 5.4 illustrates this process.

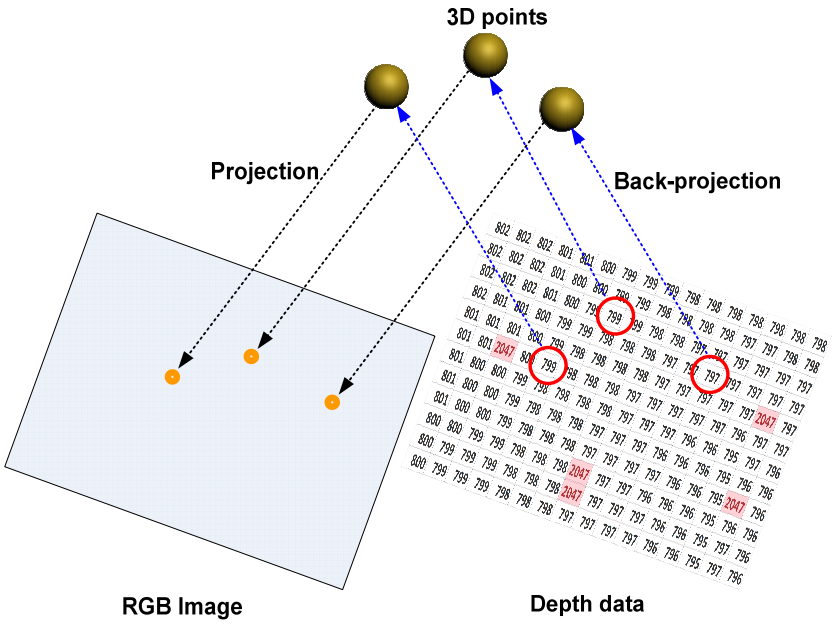


Figure 5.4: Back-projection of depth data and projecting 3D points; shaded values (red) in the depth data show unreliable raw disparities.

Given raw depth (d) values from the depth image captured by the Kinect, the world coordinates of a 3D point ($\mathbf{p} \equiv (\mathbf{p}_x, \mathbf{p}_y, \mathbf{p}_z)^T$) are calculated using

$$\begin{aligned} \mathbf{p}_x &= (xc_{x_D}) \frac{d_m}{f_{x_D}} \\ \mathbf{p}_y &= (yc_{y_D}) \frac{d_m}{f_{y_D}} \\ \mathbf{p}_z &= d_m \end{aligned} \tag{5.7}$$

where x and y denote a position in the depth data. d_m is obtained using a conversion from raw depth values (d) to depth in metres due to [273], which is

required because $\mathbf{R}$ (5.3) and $\mathbf{T}$ (5.4) are defined in metres, calculated as:

$$d_m = 0.1236 \times \tan\left(\frac{d}{2842.5} + 1.1863\right) \tag{5.8}$$

The depth image returned by Kinect may sometimes include unreliable values (those having the maximum raw disparity value of 2047 [270], as illustrated in Figure 5.4) due to reflections from corners, transparent surfaces or simply distances beyond the sensor's range. Such values are discarded from the list and the remainder stored. This list of 3D points will be provided to the plane extraction algorithm.

5.2.2 Extraction of planes

From the list of points obtained by back-projecting depth values, the main points (*i.e.* the points that will form the triplet to define a plane, $\langle \mathbf{p_0}, \mathbf{p_1}, \mathbf{p_2} \rangle$) are selected randomly. Using this triplet, two important checks are performed in order to decide on their validity for creating a plane according to the assumptions behind the explicit definition, which is their non-coincidence and non-collinearity, as mentioned earlier.

The first check is to determine whether or not the triplet has been used before to create a plane; as this may be the case even though the probability of selecting 3 points from a list of $\sim 300,000$ points is quite low. This part is efficiently implemented using a map of these triplets in which used ones are marked.

Second and more importantly, the assumptions for plane definition are checked. One of these assumptions is the distinctiveness, non-coincidence, of the 3 main points which is easily handled by checking their indices. The second assumption is the non-collinearity of main points. This is controlled by checking whether the

magnitude of the cross product of the vectors $(p_1 - p_0)$ and $(p_2 - p_0)$ is zero or not. A magnitude of zero suggests that the points lie on the same line and hence cannot define a plane.

Once a triplet satisfying these conditions is found, then a plane is created using them. This candidate plane is checked against previously found planes for similarity. This operation is required because the algorithm can find two planes on the same set of points with their normal vectors facing opposite directions or several planes with the same plane parameters found by different triplets lying on that plane.

The normal vector of a plane is checked for parallelism against the set of previous planes. For two planes (P_1, P_2), we say that the planes are parallel if the angle θ between their normal vectors (n_1, n_2) is 0 or π where

$$\theta = cos^{-1}\left(\frac{n_1.n_2}{\parallel n_1 \parallel \parallel n_2 \parallel}\right) \tag{5.9}$$

The new plane is rejected if a similar (parallel) plane is found in the set.

A final check is performed in order to obtain significantly larger planes. For this purpose, a list of 3D points is stored along with the plane's minimal definition (*i.e.* using d, n). When a point satisfies the plane equations it is added to this list and a plane should have a considerable number of points ($\tau \geq 500$) to be accepted as a valid plane otherwise it will be rejected.

5.2.3 Re-projection

After the list of planes is found from the 3D point data obtained from Kinect's depth sensor, these planes are projected onto the RGB image for display using the transformation and distortion parameters for the Kinect sensor described in

section 5.1. For a 3D point $\mathbf{p}$, first the transformation parameters ($\mathbf{R}$ and $\mathbf{T}$) are applied to $\mathbf{p}$ to find $p' \equiv (p'_x, p'_y, p'_z)^T$:

$$p' = \mathbf{R}\mathbf{p} + \mathbf{T} \tag{5.10}$$

from which the projected coordinates (p'_x, p'_y) are calculated (p'_z is ignored since the point is in image coordinates) using

$$\begin{aligned} p'_x &= \tfrac{p'_x}{p'_z} \\ p'_y &= \tfrac{p'_y}{p'_z} \\ p'_z &= \tfrac{p'_z}{p'_z} = 1 \end{aligned} \tag{5.11}$$

The radial (k_1, k_2, k_3) and tangential (p_1, p_2) distortion parameters of the RGB camera are applied to get $p'' \equiv (p''_x, p''_y)^T$:

$$\begin{aligned} p''_x &= p'_x \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) + 2p_1 p'_x p'_y + p_2 \left(r^2 + 2{p'_x}^2\right) \\ p''_y &= p'_y \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) + p_1 \left(r^2 + 2{p'_y}^2\right) + 2p_2 p'_x p'_y \end{aligned} \tag{5.12}$$

where $r = \sqrt{{p'_x}^2 + {p'_y}^2}$. Finally, the pixel locations are obtained using the camera matrix $\mathbf{K_C}$:

$$p''' = \begin{bmatrix} p'''_x \\ p'''_y \end{bmatrix} = \mathbf{K_C} \begin{bmatrix} p''_x \\ p''_y \end{bmatrix} \tag{5.13}$$

Having described the details of the plane extraction algorithm, the following section will present a method to improve its performance.

5.2.4 Performance improvement using parallelism

Algorithm 2 processes a large list of ($640 \times 480 \simeq 307200$, as there can be invalid depth values from the sensor which are ignored) 3D points, trying all possible combinations that can define a plane in the 3D space satisfying the minimum number of points and similarity conditions. Hence, it is difficult to expect a near real-time performance using this sequential algorithm.

For this reason, OpenMP [274] was used to parallelize the algorithm to achieve better performance. Initially, a trivial optimization seemed to be using a *parallel for* structure for the loops traversing the list of points and performing calculations in the algorithm. However, it did not produce a satisfactory speed-up.

Following the suggestions in [275], the parallelization strategy was modified to use a different thread to search for a plane in the dataset. Several threads were used to search for planes (as shown in Figure 5.5). When several candidate planes are found, they are checked for similarity using the method in section 5.2.2. At this stage, if similar planes are found, the planes with the highest number of points lying on them are selected. This method resulted in a significant speed-up in execution time.

The results of the sequential algorithm will be presented in the following section along with the speed-up obtained using parallelism described in this section.

5.2.5 Plane extraction results

The plane extraction algorithm presented in section 5.2 extracts planes given a depth image. After the planes are found, the points lying on these planes are projected onto the RGB image. Then, the convex hull of these projected coordinates is found and displayed along with the plane points. Figure 5.6 shows

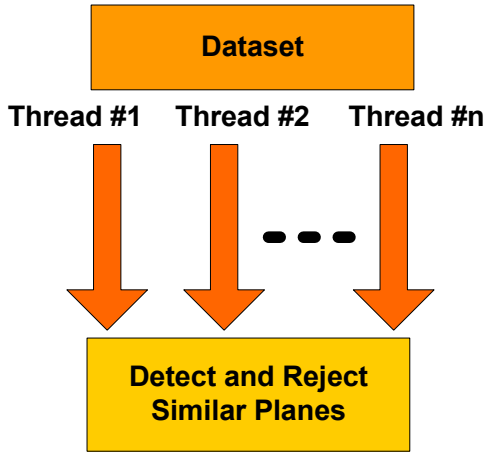
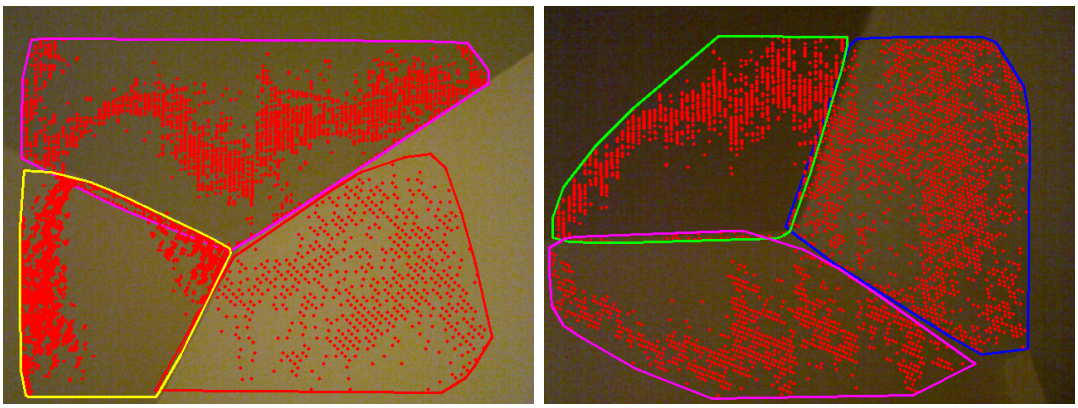


Figure 5.5: The algorithm uses a number of threads to perform a faster search for planes in the dataset.

the extracted planes for two different depth images.

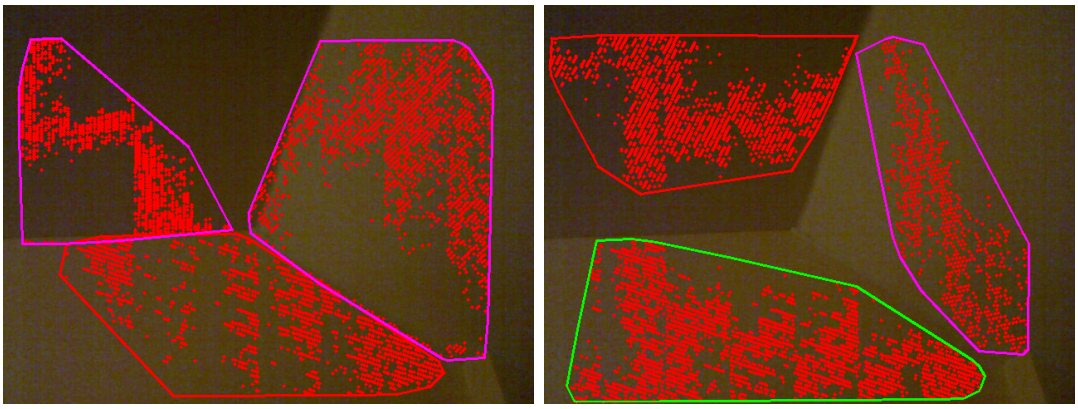


(a) Dataset 1 (b) Dataset 2

Figure 5.6: Extracted planes from the two datasets

Due to the random selection of triplets in the algorithm and the variations in the depth data from the Kinect sensor, the extracted planes can differ from execution to execution. This situation is shown in Figures 5.7(a–b) for the first and second images of Figure 5.6 respectively.

Initial execution times before adding parallelism are given in Figure 5.8 for



(a) Different planes extracted (b) Different planes extracted

Figure 5.7: Extraction of different planes from the second dataset due to the random nature of the plane extraction algorithm

different trials. A minor speed-up is visible by using the rejection of similar planes as this avoids further calculations to find the set of points lying on the plane.

Improved timing values with parallelism in different parts of the code are given in Figure 5.9. An average speed-up of $2.3 : 1$ is achieved using several threads to search for planes.

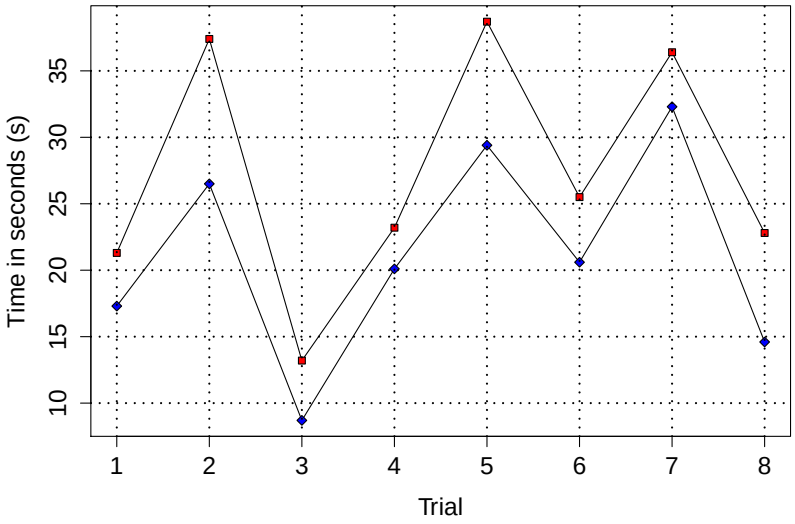
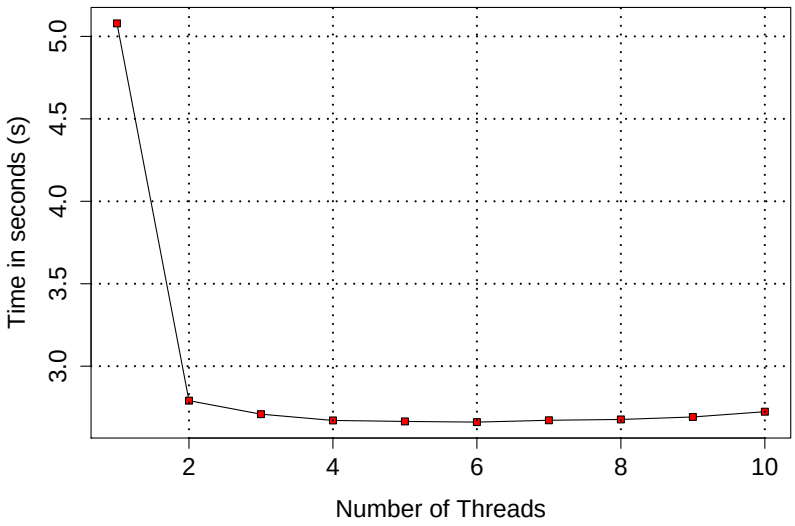
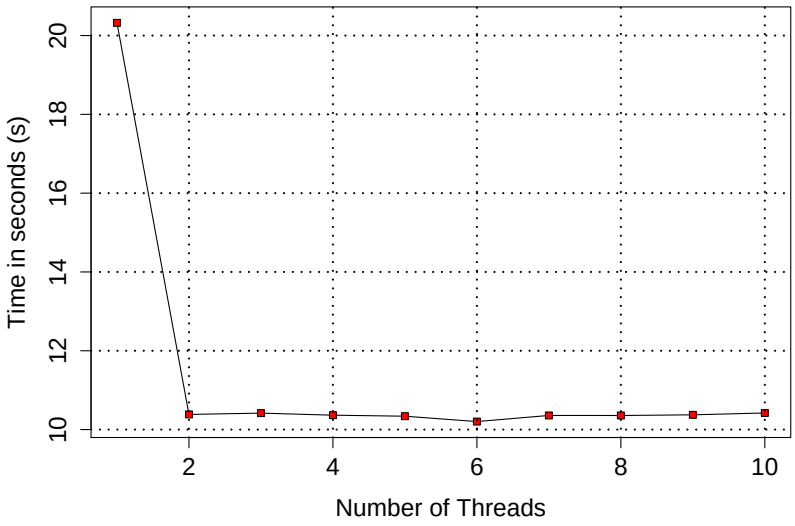


Figure 5.8: Execution time of the plane extraction algorithm for different trials using the sequential method in Algorithm 2. The blue points show the execution times before the rejection mechanism was used. Red points show the speed-up when rejection is used to eliminate similar planes in order to avoid further calculations.



(a) Depth image of 320×240 pixels



(b) Depth image of 640×480 pixels

Figure 5.9: Timing results for different sized images when different number of threads were used.

The parallel design presented here provides a good example of using the hardware resources to improve performance following the suggestions given in section 2.9. The following section presents an alternative approach which resulted in better performance for real-time.

5.3 Detecting Features for *In Situ* Augmentation

The performance of the plane extraction algorithm described in section 5.2 was far from real-time, even for a simple application. For this purpose, rather than trying to extract planar features from a large set of 3D points, the approach described here used was matching features from the environment in order to track the camera pose and use simple rectangle detection from the RGB images [11]. The reason behind this approach is that, irrespective of whether columns are cylindrical or fluted, their projection in the 2D image will be rectangular, and such shapes are relatively easy to detect.

5.3.1 Storing 3D–2D correspondences

The 3D points resulting from the back-projection of the depth sensor (section 5.2.1) and their corresponding projection points (section 5.2.3) are stored in a data structure once they are calculated. This data structure, shown in Table 5.2, allows the 3D position ($\mathbf{p}$) corresponding to a 2D pixel (p''') to be obtained quickly.

Efficiency is important when accessing this map of correspondences since it will be used for two purposes: estimating camera pose and finding the coordinates of detected rectangles in 3D.

Table 5.2: Data structure to store point data

Index	Data
$\langle p'''_{x_0}, p'''_{y_0} \rangle$	$\mathbf{p_0}$
$\langle p'''_{x_1}, p'''_{y_1} \rangle$	$\mathbf{p_1}$
$\vdots$	$\vdots$
$\langle p'''_{x_n}, p'''_{y_n} \rangle$	$\mathbf{p_n}$

5.3.2 Finding camera pose

The camera position must be found relative to the 3D points obtained from the Kinect sensor, and that is best performed by tracking reliable image features over several frames. The set of points used for tracking needs to be stable in order to achieve robust localization of the camera.

To achieve this, the FAST feature detector [55] was used to find features in the RGB image. This detector produces features that are repeatable [56] and widely scattered across the image [7], important characteristics if the resulting homography matrix is to be accurate (see the discussion in section 3.6), and fast if the system is to operate at video rate. The keypoints obtained from FAST were described using the BRIEF descriptor [64] (described in section 2.1.2), which is also known to be robust and operate at video rate. The binary structure of the BRIEF descriptor allows two descriptors to be matched using *XOR* instructions, and is therefore rapid to execute.

After the initial set of features has been obtained, these points are matched against the features detected in subsequent frames, outliers being rejected using RANSAC [71]. 3D information for matched points is obtained from the data structure (Table 5.2) alluded to in the previous subsection. There can be cases where a pixel position may not have an associated 3D entry in the table, perhaps

due to reflection of the IR beam or an alignment problem because of the disparity between the depth sensor and camera; when this happens, the closest 2D point in the data structure is used and the corresponding depth calculated as the mean of depths within a 21×21-pixel region.

Algorithms for calculating the position and orientation of the camera are well-established [276], [78] (see section 2.1.3 for more detail), provided that the intrinsic parameters (focal length *etc.*) are known through calibration. Correspondences between 3D points and their 2D projections are used in order to recover the camera pose. The corresponding 2D and 3D points stored in Table 5.2 are used to calculate the camera position using a recent PnP solution known as E-PnP [89], [88] as the initial estimate.

As discussed in [78] and mentioned in section 2.1.3, PnP solutions are easily affected by noise, and this manifests itself as an unpleasant jittering of the camera position and hence rendered imagery. To reduce this, the estimate of the camera pose is filtered. As will be shown below, a sliding window filter of size 15 reduces this effect but does not eliminate it, while a KF [93], described in section 4.2.1, was found to be more effective. The state of this filter comprises the x, y, z coordinates of the camera position and their velocities V_x, V_y and V_z. Measurements of (x, y, z) are obtained from the E-PnP algorithm discussed above. The velocities were initialized to 0.5 $units/frame$ for the three axes, reasonable for a user standing and observing ancient ruins (*i.e.,* little or no motion). The transition matrix F is

given as

$$F = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.14}$$

At each frame, first the transition matrix F is applied to the current camera position as the prediction step (Δt is the time passed between two consecutive frames). Then the measurements for the camera pose calculated using E-PnP are used to refine this prediction for updating the filter, and the state is used for the viewpoint of augmentation. It was found that this filter reduced jittering to the point where it was imperceptible.

5.3.3 Detecting objects for augmentation

After determining the camera pose, the next step is to find planar objects— to be augmented, in this work, by synthetic column models as shown in the applications in chapter 7. The approach presented here first finds rectangular features in the scene using the RGB images instead of the 3D planes (section 5.2). The detected rectangles were filtered based on their sizes and orientations. Finally, a PF was used to track the extracted rectangles since some of them could not be detected in all frames.

For the RGB image, any visual noise was reduced by Gaussian smoothing [277]. Then the Canny edge detector [278] was applied to each of the colour channels independently. Contours were extracted from the edge images using an approxi-

mation method [279]; contours that contain four vertices with angles between pairs of lines joining these vertices close to 90° form reliable rectangles as depicted in Figure 5.10. Small rectangles and rectangles having inappropriate aspect ratios (vertical columns are required for this application) were rejected.

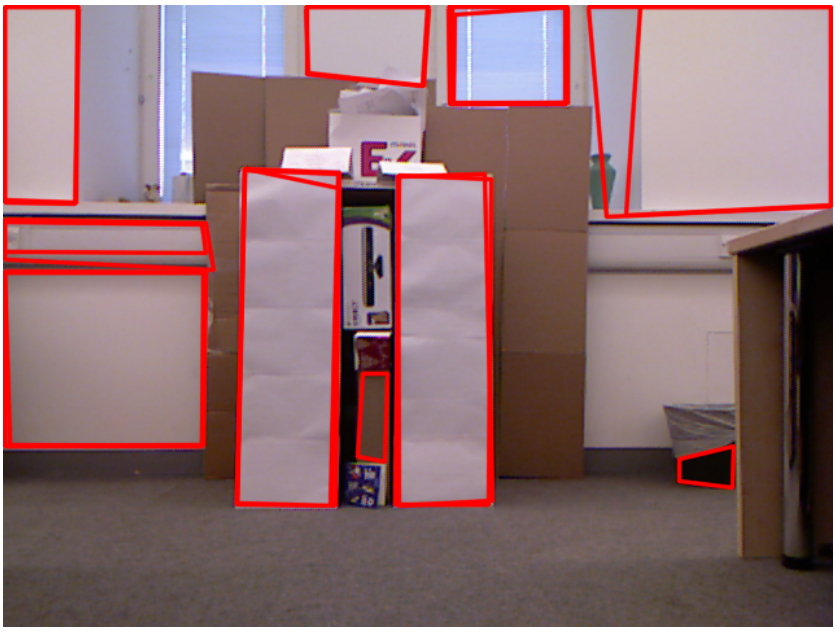


Figure 5.10: Rectangular features located within an image

The rectangles detected using the method described above tend to disappear and re-appear from frame to frame due to the changes in lighting conditions. For this reason, the centres of the rectangles were tracked using the *Condensation* algorithm [102] described in section 4.2.2.

The state of the filter is initialized with the centre coordinates of a detected rectangle and a set of particles were initialized randomly within the 20-pixel radius of the centre. In the following frames, each time the same rectangle is detected, the centre coordinates of the newly detected rectangle (still the same rectangle tracked with small variations in the centre location) are used to update the particle

weights. The number of particles for the algorithm was selected as $N = 50$, which was found to be sufficient (*i.e.* not causing particle deprivation [94]). At each frame, the particles are updated using the result of the rectangle detection algorithm described above, resulting in the centre of the rectangle being tracked robustly as depicted in Figure 5.11. After the re-sampling update is performed, the particle with the largest weight is selected both for augmentation and propagation into the next state.

The 3D position of each rectangle's centre, tracked by the PF mentioned, will be retrieved from the map in Table 5.2 and column models will be rendered at these 3D positions as an interesting AR application in chapter 7.

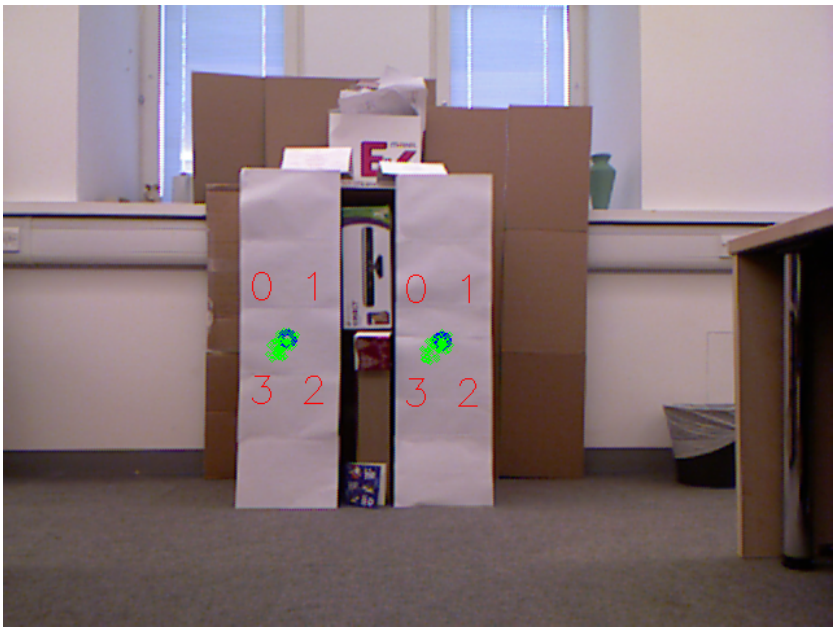


Figure 5.11: Tracking selected rectangles. Small green circles show the estimates of their centres whereas the large dark blue circle is the particle with largest weight.

The complete augmentation algorithm is given in Algorithm 3.

Algorithm 3 *In-situ* augmentation

Require: I_C: RGB image, I_D: Depth image, 3D models for augmentation.
  Load models and camera calibration parameters.
  Initialize the KF and rendering environment.
  for all frames **do**
    if first frame **then**
      Extract the initial set of features for tracking using FAST detector and BRIEF descriptor.
      Find the initial number of rectangles for augmentation, initialize the PF.
    else
      Compute 3D points from the depth data.
      Calculate projections and create the point map.
      Find feature matches by detecting, describing and matching features from the RGB image for the new frame.
      Calculate camera pose using E-PnP.
      Update the KF for camera pose using measurements and set viewpoint for augmentation.
      Detect rectangular objects.
      Update PF.
      Retrieve centre coordinates for augmentation.
      Render the view.
    end if
  end for

5.3.4 Performance of the *in-situ* augmentation algorithm

The errors in the initial estimation are shown in Figure 5.12. The relative error is calculated for the initial set of 3D–2D correspondences by re-projecting 3D points using the estimated translation and rotation for the camera, while the true re-projection error is calculated using ground truth, obtained from the knowledge that the camera is stationary. A mean difference of $\simeq 0.26$ pixels was obtained between the true and estimated re-projection errors, showing that the estimation is reasonably accurate.

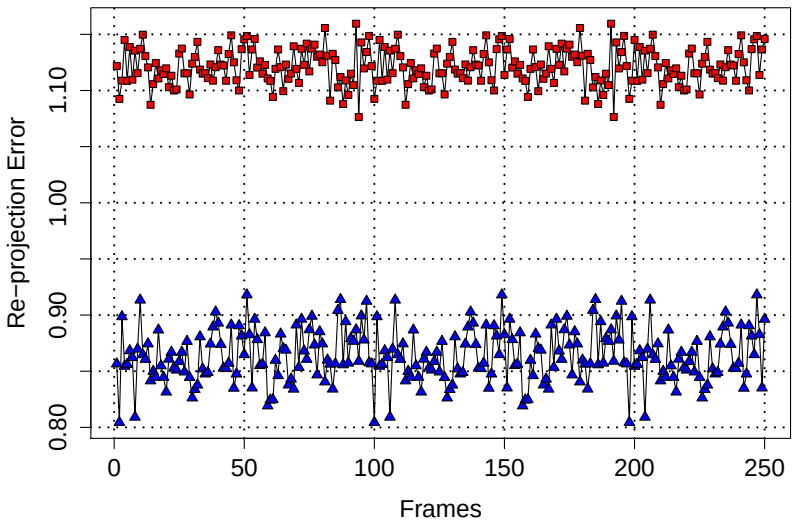


Figure 5.12: Estimated (blue) and true (red) re-projection errors calculated with E-PnP

Again using the ground truth, the calculated errors in rotation and translation and are given in Figure 5.13. Calculating the rotation error involves converting the actual and estimated rotation matrices into quaternions and finding the distance between the two quaternions. For the translation error, the Euclidean distance

is calculated between the actual and estimated translations. Fluctuations are substantially higher for the translational error ($\sigma_{t_{error}} = 0.009$) compared to the rotational error ($\sigma_{r_{error}} = 0.0004$). As alluded to above, the magnitude of this translational error results in the rendered augmentation being unstable ('jittering').

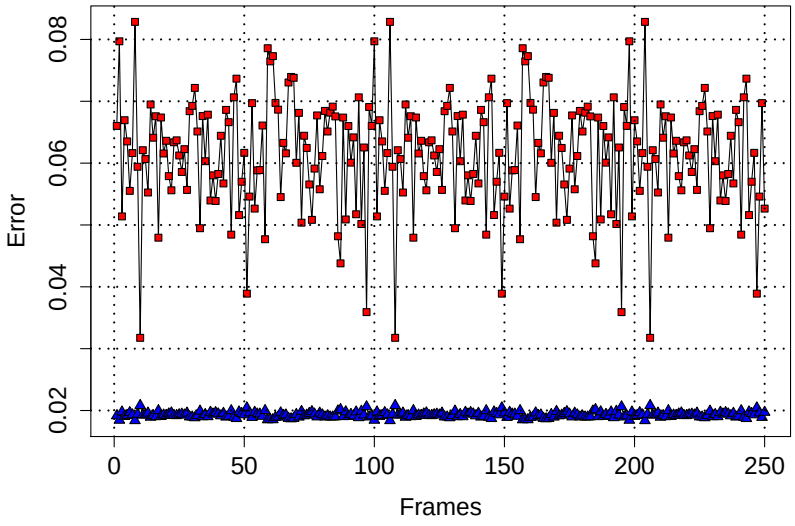


Figure 5.13: Rotational (blue) and translational (red) errors for the camera position estimate

Figure 5.14 shows the results from sliding window and Kalman filters, and it is clear that the Kalman filter produces more stable positions. The jitter is reduced from $\bar{\sigma}_{raw} = 0.43$ to $\bar{\sigma}_{Kalman} = 0.07$, where $\bar{\sigma}_{raw}$ and $\bar{\sigma}_{Kalman}$ are the standard deviations of the camera coordinates for the initial estimate and Kalman filtered results.

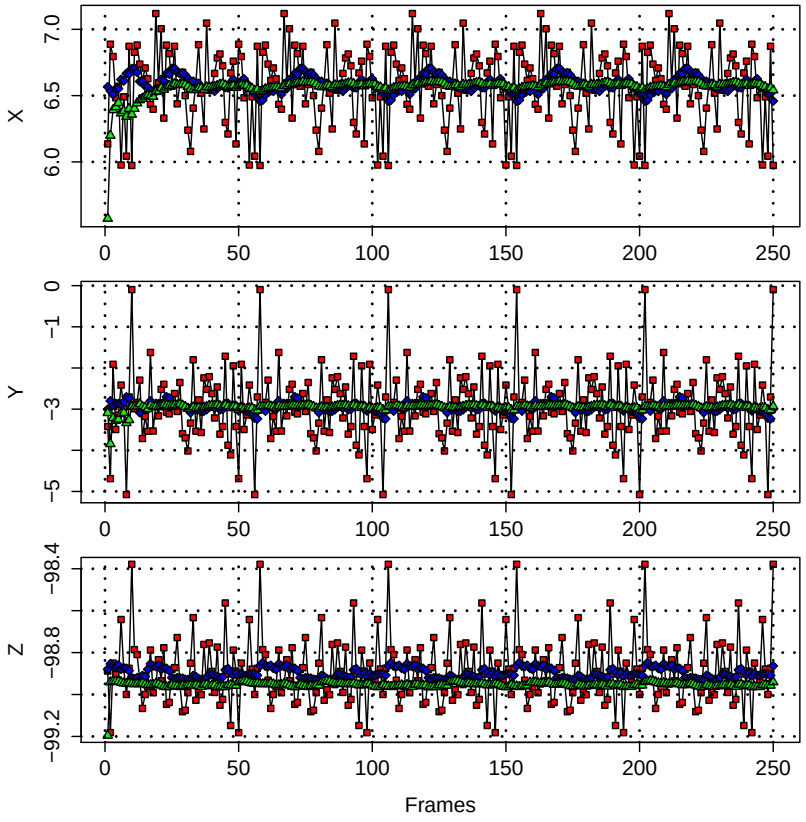
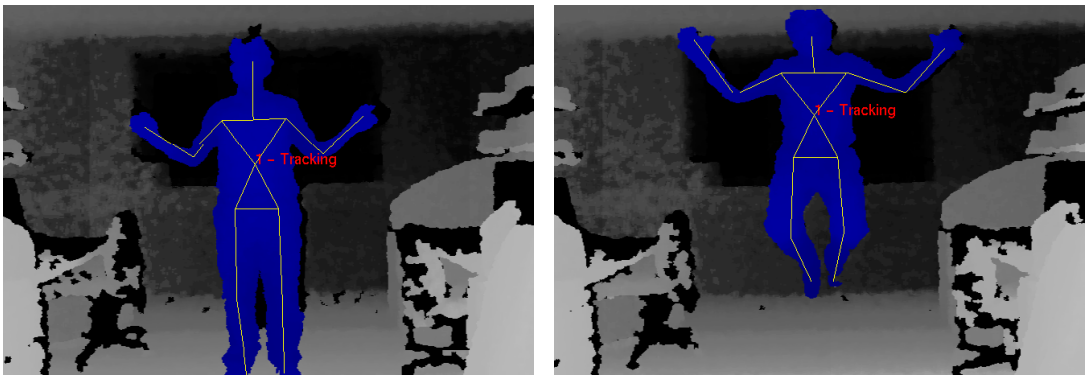


Figure 5.14: Camera coordinate estimations using raw estimation result (blue), sliding window (red) and Kalman filter (green)

5.4 Identifying Body Parts

Another interesting application of Kinect is due to its human-skeleton tracking capabilities which is done by detecting the user in the depth image and then finding joint positions by inferring the skeleton pose using a decision forest [36] as explained in section 2.1.1.

The Open Natural Interaction (OpenNI) NITE library [280, 281] processes depth information from Kinect and performs detection of body parts as shown in Figure 5.15.



(a) A standing user in the calibration pose—see discussion in the text

(b) A jumping user

Figure 5.15: User tracking with Kinect

User tracking can be accurately performed if the user is standing in front of the Kinect sensor (2.5m is given as an ideal distance) with his/her body (the upper body in particular) inside the FOV and not occluded [281].

In the initialization of the skeleton tracking algorithm, a calibration is automatically performed for improved accuracy in the user position. At this stage, the user is expected to stand in the calibration pose, referred to as "Psi" (Ψ), as shown in Figure 5.15(a). Once, the user pose is obtained successfully, then

each joint's position and orientation can be retrieved. It is also noted that the accuracy for joint positions is higher than for orientations [281]. A confidence value is provided by the library as an indicator of how well the tracking algorithm is progressing.

From the 31 joints that can be tracked with Kinect, three joints were used in the thesis: the head, torso and right hand as shown in Figure 5.16.

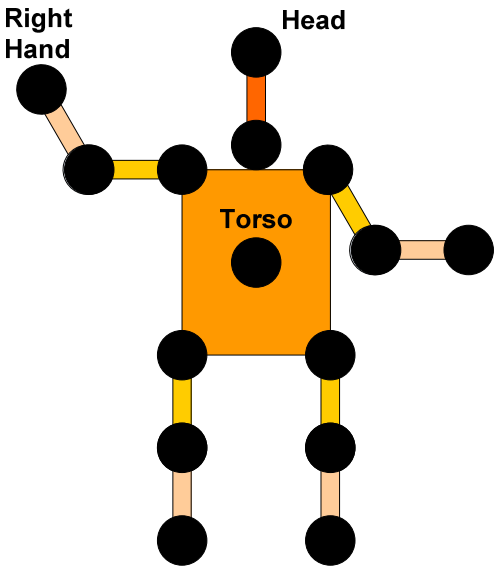


Figure 5.16: Skeleton joints identified by OpenNI

As each joint is recognized by the library, its position and orientation are returned. These transformations are absolute, not relative, and so can be used directly for rendering, with the exception that the rotation matrix for orientation must be converted to a rotation vector using Rodrigues' formula given in (4.38).

Sample data for the joint positions mentioned above for a user standing with some movements in all three dimensions and waving his hand is presented in Figures 5.17, 5.18 and 5.19. It can be seen that the motion of the user's body along Y (up-down) and Z (forward-backward) axes are reflected in all body parts.

The wave gesture is clear in Figure 5.19 in the X (left-right) axis.

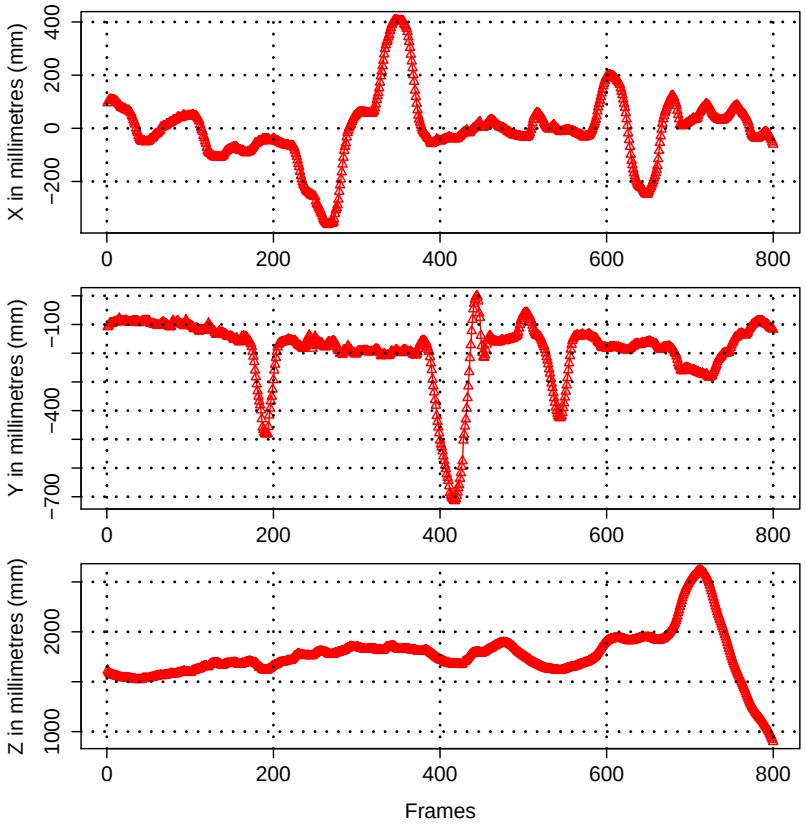


Figure 5.17: Torso coordinates

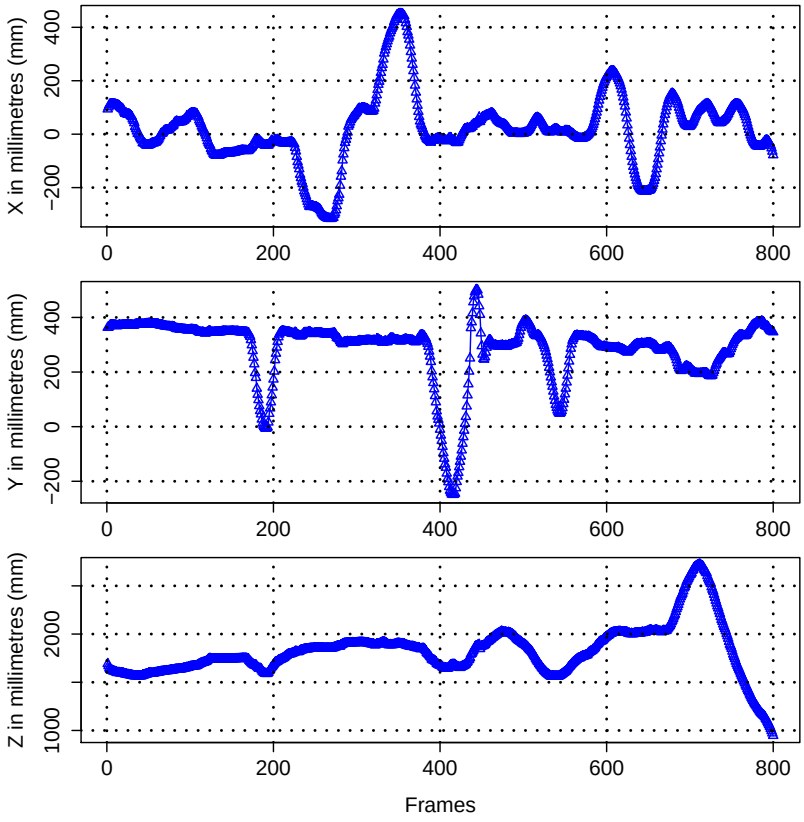


Figure 5.18: Head coordinates

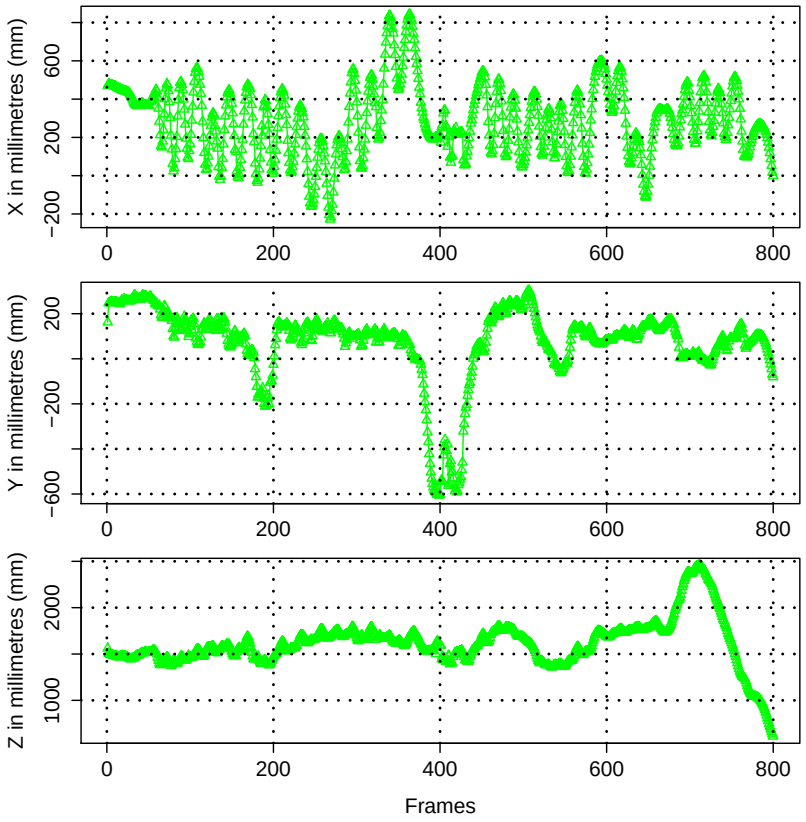


Figure 5.19: Right-hand coordinates. Note the change in X coordinate of the hand due to the waving motion

This skeleton tracking capability will be used for augmenting participants with a *galea* (Roman helmet), toga and sword in an AR application in chapter 7.

5.5 Remarks

The use of Microsoft's Kinect sensor for finding planar features in the environment and its skeleton tracking features were investigated in this chapter. The discussion started with calibration process for the Kinect which is required for both finding the intrinsic parameters of both RGB and depth sensors and finding the transformation between the two sensors in order to align their output images.

Having found the calibration parameters, an algorithm for extracting planar features using only data from Kinect's depth sensor was presented. The algorithm selected three main points randomly in order to define a plane using a normal vector and a distance parameter (explicit definition). Similar planes (*e.g.* planes having parallel normal vectors) are rejected by the algorithm. Then, a plane is stored if it has a considerable number of points that satisfy its equation. Projections of the plane points on the RGB image were shown with a convex hull surrounding them as the resulting planes.

The algorithm tried every possible combination from a large set of 3D points. While this resulted in finding large planar structures in the environment, random search in such an enormous search space resulted in an increased time-cost. Attempts to use parallelism for searching for planes significantly reduced the execution time, though the timing was still not satisfactory to be used in an real-time application.

Another algorithm to overcome this problem was proposed, finding rectangular features considering the shapes of column projections in 2D images. This

approach first described a method for finding the camera pose using the 3D–2D correspondences obtained from the Kinect sensor. A special data structure was defined for storing and retrieving these correspondences efficiently. Later, these correspondences were used with a recent solution of the PnP problem in order to get an initial estimate for the camera pose. Due to noise from both sensors of Kinect, jittering was observed which manifested itself in a shaky output in rendering. This was first tackled by a simple sliding window filter which reduced the jittering and then completely solved using a KF.

In order to find the actual features in the scene that will be used for *in situ* augmentation, a rectangle detection and tracking method was presented. This approach extracted contours from edge images and then filtered the extracted rectangles for a specific size and aspect ratio. The 3D coordinates were obtained from the data structure defined earlier constructed using the depth data from Kinect. These rectangles were tracked using a PF for additional stability in the application since they tend to disappear and re-appear in some frames.

Finally, the chapter used human skeleton tracking features of the Kinect sensor and OpenNI NITE library for tracking several joints of users. The results provided here for both skeleton tracking and finding features in the environment for augmentation along with finding camera pose has shown how a depth sensor can facilitate challenging tasks. The approaches described here will be used for two interesting applications in chapter 7.

Due the brightness and shadows resulting from direct sunlight, it is not always possible to use Kinect outdoors. For this reason, use of other sensors will be investigated along with the vision-based motion estimation method of chapter 4 in the following chapter.

CHAPTER 6

FUZZY INTEGRATION OF MULTIPLE SENSORS

A tracking system that will be used for AR applications has two main requirements: accuracy and frame rate (*i.e.* fps). The first requirement is related to the performance of the pose estimation algorithm and how accurately the tracking system can find the position and orientation of the user in the environment. Accuracy problems of current tracking devices, considering that they are low-cost devices rather than the tactical-grade sensors used in military operations, cause static errors during the motion estimation process. The second requirement is related to dynamic errors (the end–to–end system delay; see section 2.8) occurring because of the delay in estimating the motion of the user and displaying images based on this estimate. It is known that the human visual system can process 8–10 images in a second and current industrial standards for frame rate is between

25 and 30fps [282].

The vision-based user tracking method described in chapter 4 can provide about four motion estimates in a second which is not fast enough to capture every important motion the user may have performed. In addition to this, the frame rate obtained using the vision-based method as the only estimate is not high enough to provide satisfactory results in the AR application considering the additional time required for rendering and displaying the models. Finally, this motion estimate has an increasing error due to the dead-reckoning approach followed.

For the reasons described above, a sensor fusion approach was followed to provide more accurate estimates of motion more quickly using the high sampling rates of an IMU and GPS. Integration of data from GPS and IMU sensors has been well-studied [283] in order to improve upon the robustness of the individual sensors against a number of problems related to accuracy or drift. The KF is the most widely-used filter due to its simplicity and computational efficiency [284].

The literature also presents attempts to combine vision with other sensors. For instance, [210] used stereo cameras with a low-cost GPS receiver in order to perform vehicle localization. SIFT was used for feature matching in a sub-mapping approach. Armesto *et al.* [285] used a fusion of vision and inertial sensors in order to perform pose estimation for an industrial robot by using the complementary characteristics of these sensors (see section 2.7.1). GPS position was combined with visual landmarks (tracked in stereo) in order to obtain a global consistency in [202]. A similar approach was followed by Agrawal *et al.* [209] on an expensive system using four computers. Bleser [198] combined vision-based motion estimates with IMU in a PF framework for AR in indoor environments. In [286], a similar approach was followed in order to perform localization for an UAV: vision (a camera facing downwards) and inertial sensing was used together in a PF for

position and orientation estimation in 2D. For estimating the altitude, a pressure sensor was used.

Recently, Oskiper *et al.* [287] developed a tightly-coupled EKF visual–inertial tracking system for AR for outdoor environments using a relatively expensive sensor (XSens, MTi-G, given in section 6.1.3). The system used feature-level tracking in each frame and measurements from the GPS in order to reduce drift. In addition to this, a digital elevation map of the environment was used as well as a pre-built landmark database for tracking in indoor environments where GPS reception is not available (although it was claimed that no assumption about the environment was made). The error was found to be 1.16 metres.

Attempts to improve the accuracy of the filtering have also been made using adaptive approaches. In some studies, values for the state and measurement covariance matrices were updated based on the innovation [288] and recently fuzzy logic was used for this task [289, 290]. Another approach for fusing accelerometer and gyroscope for attitude estimation is also based on fuzzy rules [291] in order to decide which of the accelerometer or the gyroscope will be given weight for estimation based on observations from these sensors such as whether a mobile robot is rotating or not. A later approach [292] used the error and dynamic motion parameters in order to decide which sensor should have a dominant effect on the estimation.

Some other studies suggest [86] or use [87, 293–295] the idea of employing different motion models for recognizing the type of the motion for two-view motion estimation and visual SLAM. Different studies [293–295] used geometric two-view relations such as general, affine or homography in order to fit these models to a set of correspondences and using the outliers for obtaining a penalty score in a Bayesian framework.

Civera *et al.* [87] used a bank of EKFs in order to apply different motion models to several filters concurrently and select the best model in a probabilistic framework. This approach incorporated 3 motion models, namely stationary, rotating and general, separating models for motions including translations and rotations.

The tracking system developed in this chapter takes a different approach by combining estimates from a camera, a low-cost GPS and a low-cost IMU for better accuracy in motion estimation. A second novel contribution presented here is employing fuzzy logic to choose the best-fitting of several possible motion models, ensuring that the filter state is more consistent with the measurements and hence converges faster. Furthermore, the design here does not bring additional computational burden due to the simple design and efficient implementation of the rule-base.

The rest of the chapter starts by describing the low-cost sensors, GPS and an IMU, used in the thesis in section 6.1. The working principles of these sensors are presented as well as the sources of error found in these sensors.

The sensor fusion stage uses motion estimates from three sensors; how these individual estimates are calculated are described in section 6.2. For the camera motion estimate, the two-view approach described in chapter 4 is used. The position estimate for GPS is fairly straightforward since there is a direct conversion between the reference frames used by the GPS and the coordinate system used in the thesis. For the estimate from the IMU, a recently developed filter [6] is employed for obtaining the orientation estimate where conventional double-integration was used for the position estimate.

Once motion estimates from each sensor are obtained, these estimates are then fused in a KF framework in order to provide a single output for position and

orientation in section 6.3. The design of the filter and use of several threads for improving its performance are described and then the tracking system itself will be presented.

The sensor fusion algorithm presented initially used a single motion model, ignoring any the changes in a user's motion patterns which will not be constant during operation. To ameliorate, a multiple motion model approach is proposed using a fuzzy rule-base defining the transitions between different motion models in section 6.4.

Tracking results for the developed system will be presented in section 6.5 and comparisons will be made between conventional GPS–IMU integration and this sensor fusion approach, which also uses the estimates from the vision-based algorithm. Differences in tracking results when the multiple motion model approach is used instead of a constant motion model will be presented. Finally, the chapter will be concluded in section 6.6.

6.1 Sensors

This section presents an overview of the sensors namely GPS and IMU used in the sensor fusion algorithm presented in this chapter. After presenting the main principles behind these two sensors, details of the sensors used in the experiments will be given.

6.1.1 Global positioning system (GPS)

GPS is a satellite network called *NAVSTAR* which regularly transmits encoded data, allowing a user to find his or her position on the Earth, to some accuracy. The network initially consisted of 24 satellites but now operates with more

satellites for greater accuracy and increased robustness [284]. GPS is owned by United States Department of Defense, constantly orbiting the Earth and using solar power for energy.

For non-military applications of GPS, the accuracy was initially limited to 100 metres however this was dispensed with in 2000. There is still a limitation on altitude of 60,000 feet (18.29km) and speed of 1,000 knots (1900km/h) in order to prevent it being used for missile guidance [296]. Positional accuracy typically varies between 2–20 metres.

The satellites orbit the Earth twice a day and use trilateration [283, 297] (similar to triangulation but the former uses distances rather than angles) to find the position of the receiver; a fourth satellite is required to find the position in 3D. The design of the GPS system (*i.e.* the number of satellites and their trajectories) is also based on this constraint so that at least six satellites can be visible from any user position [284] in order to provide a global coverage, even in cases of single satellite failure.

Figure 6.1 shows how trilateration works to find the position of a user. Synchronization errors between the receiver and the satellites cause an uncertainty in the calculated distance. This problem is the main reason behind the inaccurate positional values from the GPS.

Due to the low-power satellite transmission, the accuracy of GPS is affected by geographical conditions such as the structure of the terrain. These problems are related to the satellite visibility (*i.e.* line of sight) and can occur in deep canyons, under dense vegetation or tree canopy [75]. A similar problem occurs in urban environments where the visibility of the satellites are affected by rows of high buildings, creating a canyon like structure. This problem worsens in rough urban canyons where the GPS signal is received from a multipath indirect reflection [299].

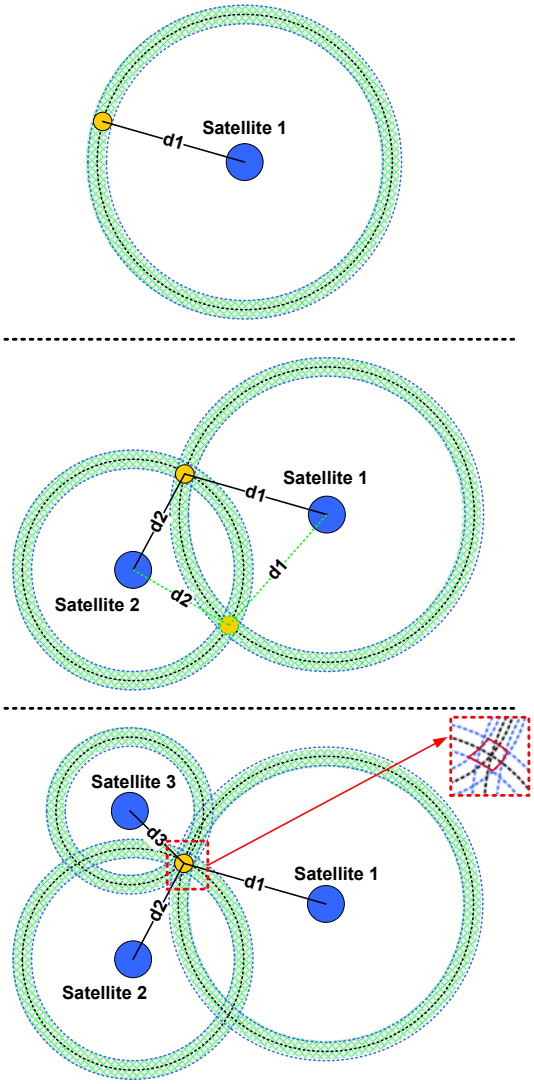


Figure 6.1: Trilateration to find position. The object to be located is shown with the yellow circle. Blue circles indicate the satellites visible by the object. (Top) Distances from the object to each satellite i are shown with d_i. (Middle) Two locations are possible when two satellites are used. (Bottom) The object can be located within a degree of error when three satellites are used. The error is due to uncertainty (shown with green areas) of distances arising from synchronization problems between the receiver and the satellites. Red-dashed region indicates possible positions of the object. Following [284, 298]

6.1.2 Inertial measurement unit (IMU)

An IMU consists of two individual sensors namely an accelerometer and a gyroscope which are used to detect linear accelerations and angular velocities respectively. The device is called a Magnetic, Angular Rate and Gravity sensor (MARG) when it also includes a compass. The working principles of these two sensors, fabricated as Micro Electro Mechanical System (MEMS) devices, is described below.

Accelerometer

Accelerometers measure acceleration by measuring the force that caused the acceleration. Figure 6.2 aims to explain the logic behind accelerometers. Considering the mechanical design, an external acceleration reflects itself as a force exerted to the body inside the accelerometer causing it to move in the opposite direction to the acceleration. This force will compress or stretch the springs holding the body. In the electrical design the force will push the moving plate so that the distance between the capacitor will change, eventually altering the voltage of the capacitor.

Gyroscope

A gyroscope is used to measure orientation. The working principle of a gyroscope for detecting a change is similar to an accelerometer since it also measures the voltage changes in capacitors as shown in Figure 6.3. As the base plane rotates, the object inside the body resonates and as a result of the Coriolis force [301] which is perpendicular to the centrifugal force, which is indeed a form of inertia as a tendency to move objects from the centre of a spinning circle, the sensors can detect the changes in voltage.

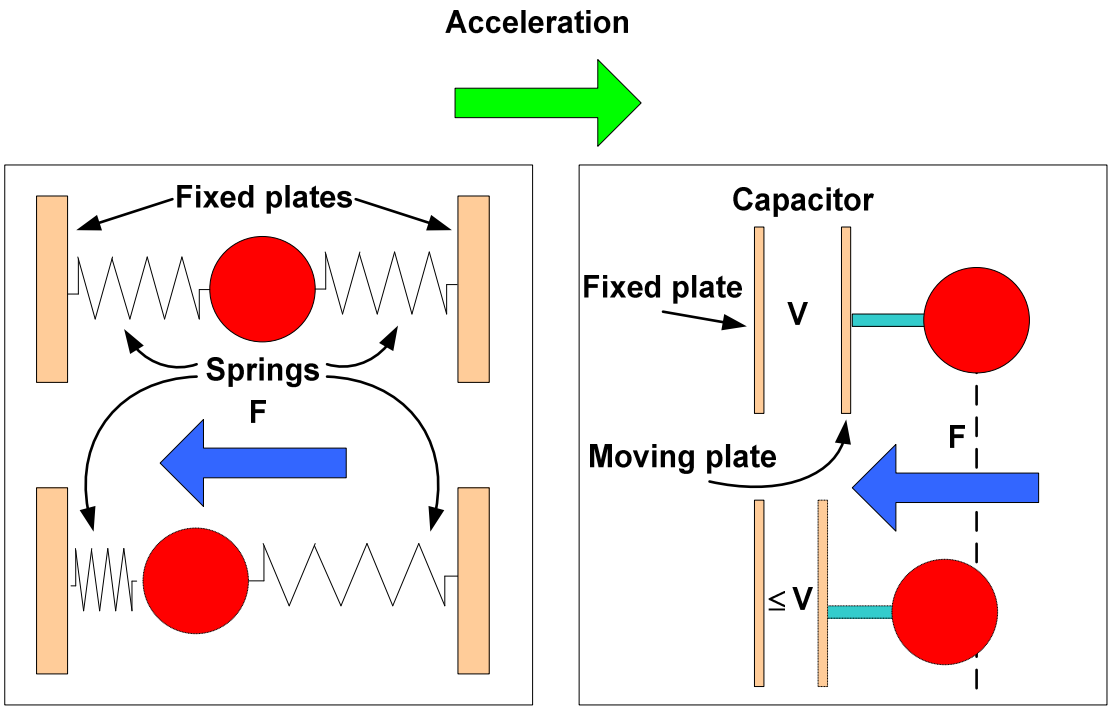


Figure 6.2: Simple diagram of two accelerometer designs. Mechanical design is shown on the left and the electrical design is shown on the left. When the accelerometer is subject to an acceleration, this acceleration will cause the body (red circle) inside it to move with an inertial force in the opposite direction. In a mechanical design this generated force will compress and stretch the springs whereas the in the electrical design the body will move one of the plates of the capacitor changing the voltage. Following [300].

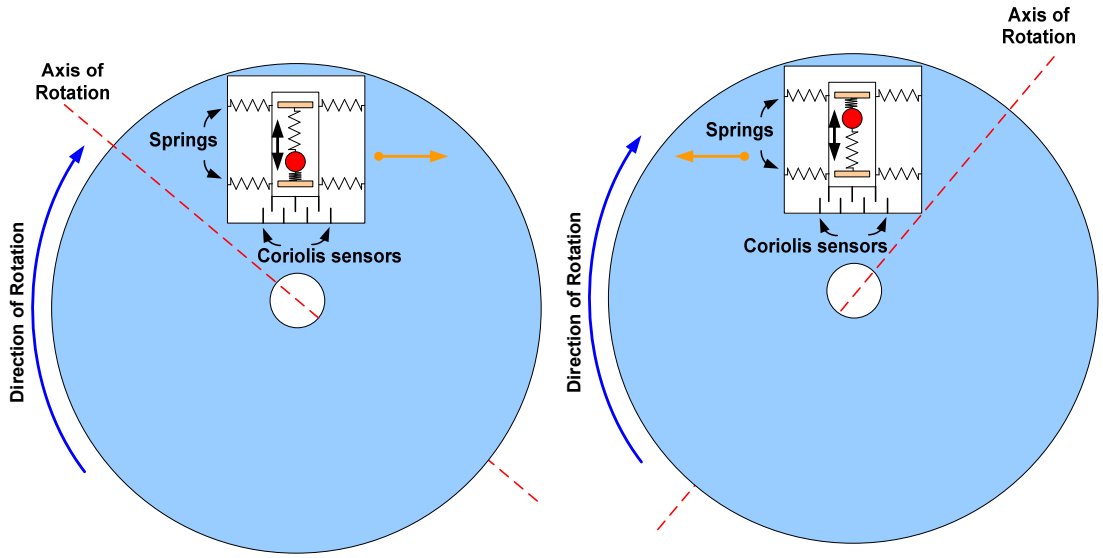


Figure 6.3: Simple diagram of a gyroscope. The Coriolis force resonates the inner frame of the sensor and causes the voltages between the fingers to change depending on the amount of rotation. Following [302, 303].

Having described the logic behind accelerometers and gyroscopes, it is important to note that current digital sensors do not directly follow the designs shown in Figures 6.2 and 6.3 mainly due to the space requirements. Instead, semiconductor chips are used to fabricate them on silicon (MEMS) [304].

Sources of error

There are sources of error which reduce the accuracy of the sensors used in inertial sensing. The most common of these errors are *accelerometer bias* and *gyroscope drift*. These can be compensated for by finding the parameters through calibration or from the product specifications and incorporating them into calculations as will be discussed in Section 6.2.3. Another source of error is related to the temperature [305]. Changes in the IMU temperature affect the bias and drift from the sensors. Vibrations can also cause noisy readings from the sensors, especially

during the calibration of the sensor in order to find the bias. To prevent this, it is essential to place the sensor on a stable surface where there is no vibration from sources such as a computer.

While the first three sources of error constitute an important concern in the system presented in the thesis, the temperature is less of a problem. (Though it can cause problems if the proposed system is to be used in a UAV [306].)

6.1.3 The sensors used in this research

There are many different sensors provided by various companies, as shown in Table 6.1; pricing changes based on the accuracy or the resolution of the sensor. Some sensors (*e.g.* MTi-G, IG500N) include an on-board KF to integrate GPS data with the estimates from gyroscopes and accelerometers in order to produce more accurate results. This is another main factor for increased cost.

Table 6.1: GPS and IMU sensors by different vendors

Company	Model	Price	Product
XSens	MTi-G	≈£3300	
SBG Systems	IG500N	≈£3100	
DIYDrones	ArduIMU+V2, U-Blox5 GPS	≈£65+ ≈£55	
Phidgets	1056 Spatial, 1040 GPS	≈£75+ ≈£50	

The GPS and IMU boards provided by Phidgets are used in the experiments. One main reason behind this was that the sensors can be directly used with a

Universal Serial Bus (USB) interface. Secondly, the sensors were relatively cheap when compared to those provided by XSens[1] or SBG Systems[2].

According to the product specifications of GPS in Table 6.2, the 1040 GPS is specified to be accurate within 2.5 metres in the best case and can deliver 10 samples per second. The sensor also include a rechargeable battery that is used to store recent positions of the sensor so that it can provide "hot-starts," though it may take a few minutes for GPS to start receiving fixes if it has not been used for a long time.

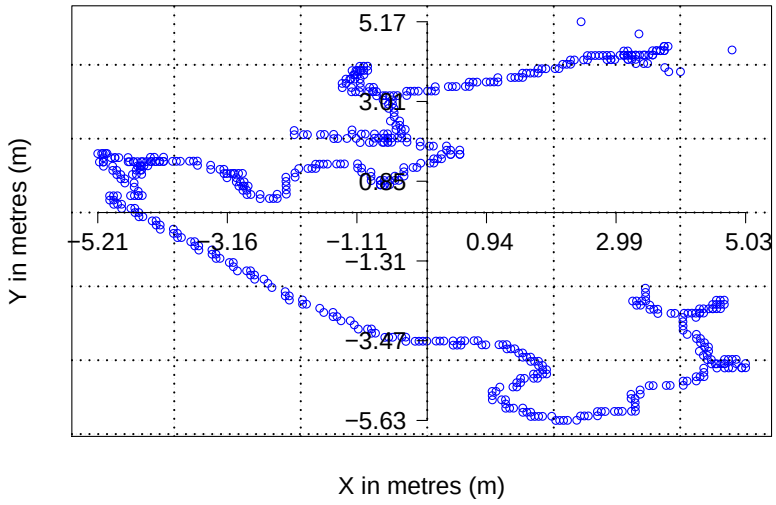
Table 6.2: Specifications for the Phidgets 1040 GPS [307]

Circular error probable	2.5m
Update rate	10 samples/s
Timing error	300ns
Re-acquisition time	1s

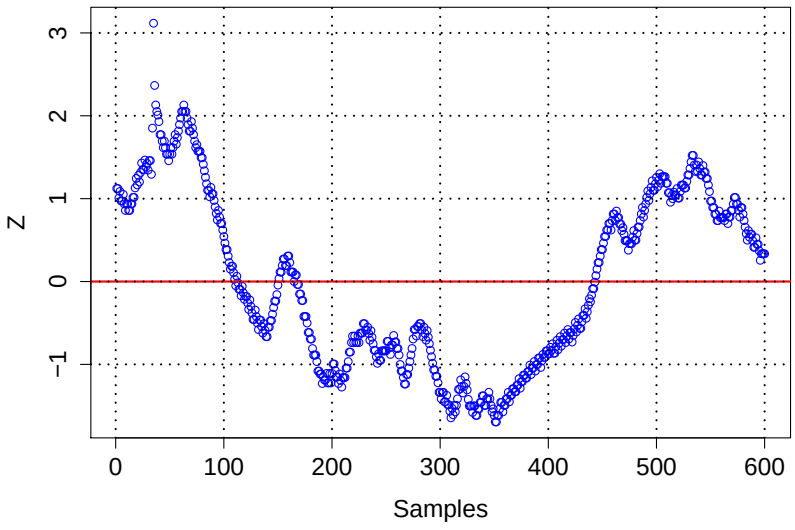
An experiment was performed with the GPS receiver left stationary in an outdoor environment with 9–11 satellites visible. The variation in the position acquired by the receiver is presented in Figure 6.4 where it can be seen that the positional error is, in fact, more than 2.5 metres.

The Phidgets 1056 IMU combines a tri-axis gyroscope, tri-axis accelerometer and a tri-axis compass (actually making the device a MARG, but the magnetic field sensor was not used in the system presented here, so the sensor is referred to as an IMU), on a single board. The product specifications for the IMU is presented in Table 6.3. The gyroscope can measure changes of 0.02 degrees per second (as shown in Table 6.3). The drift is 4 degrees per minute. The accelerometer has a resolution of 2.28×10^{-4}g. The board on which these two sensors are mounted can provide samples every 4 milliseconds (250 samples per second).

[1] `http://www.xsens.com/`
[2] `http://www.sbg-systems.com/`



(a) GPS error for X and Y coordinates



(b) GPS error for the Z coordinate

Figure 6.4: Errors in GPS data. Data is collected from a stationary location with 9–11 satellites visible. For the first plot, the centre is the mean of 600 sample readings. Points show the distances from the mean. Second plot is displaying the changes in altitude readings.

Table 6.3: Specifications for the Phidgets 1056 IMU [308]

Gyroscope	
Resolution	0.02°/s
Drift	4°/min
Accelerometer	
Resolution	228 μg (2.2mm/s^2)
Board	
Sampling speed	4 ms/sample

An experiment to assess the drift was performed by fixing the IMU on a stable surface and storing the values from the sensor. The actual drift for the yaw, pitch and roll parameters was found to be $\simeq 5°$ per minute (while this was specified to be 4°/min in Table 6.3) as shown in Figure 6.5.

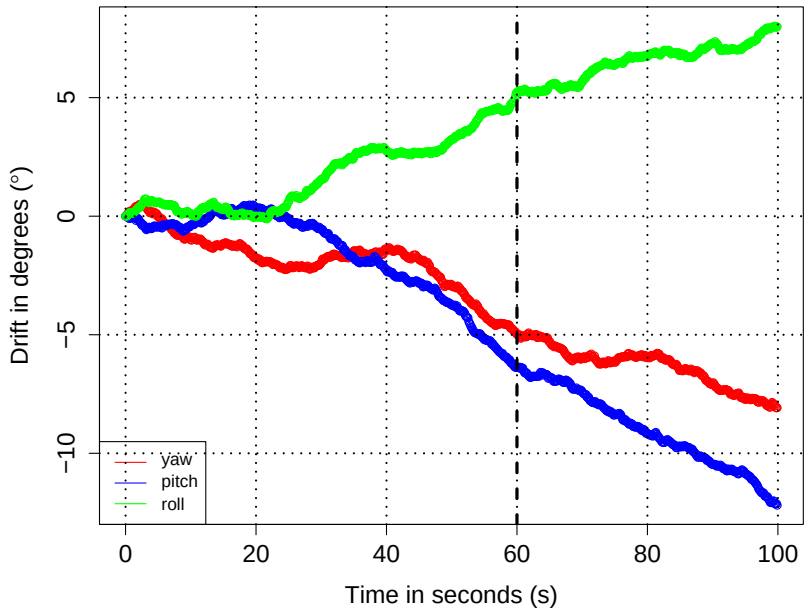


Figure 6.5: Gyroscope drift for the yaw, pitch and roll parameters. Actual drift, calculated when the IMU was left stationary on a stable surface, is found to be around 5° per minute.

These specifications (Tables 6.2 and 6.3) are essential for calculations for ob-

taining motion estimates for orientation in particular. The two sensors described here will be used for obtaining motion estimates in the following section.

6.2 Finding Motion Estimates from Sensors

Before describing the details of the sensor fusion algorithm, this section describes the methods used to obtain measurements from the camera, GPS and IMU.

6.2.1 Camera motion estimate

Chapter 4 described a vision-based user tracking algorithm providing motion estimates obtained using a two-view approach, by calculating the essential matrix between the most recent two keyframes. The algorithm extracted a new keyframe based on the number of features matched as described in section 4.4.2. The motion estimate between the keyframes (section 4.4.3) was in the form of a rotation and translation which were incorporated into a transformation matrix (Tr of (4.39)).

This transformation matrix was initially applied, following a dead-reckoning approach, to the last position estimated by the camera in order to obtain the new position. In the sensor fusion algorithm, this transformation will be applied to the estimate obtained using GPS and IMU.

6.2.2 GPS position

The data obtained from the Phidgets 1040 GPS is in well-known NMEA format and includes position, the number of visible satellites and detailed satellite information for a position P on Earth's surface, as shown in Figure 6.6.

Using this information, the GPS coordinates can be converted from geodetic latitude (ϕ), longitude (λ) and altitude (h) notation to ECEF Cartesian coordi-

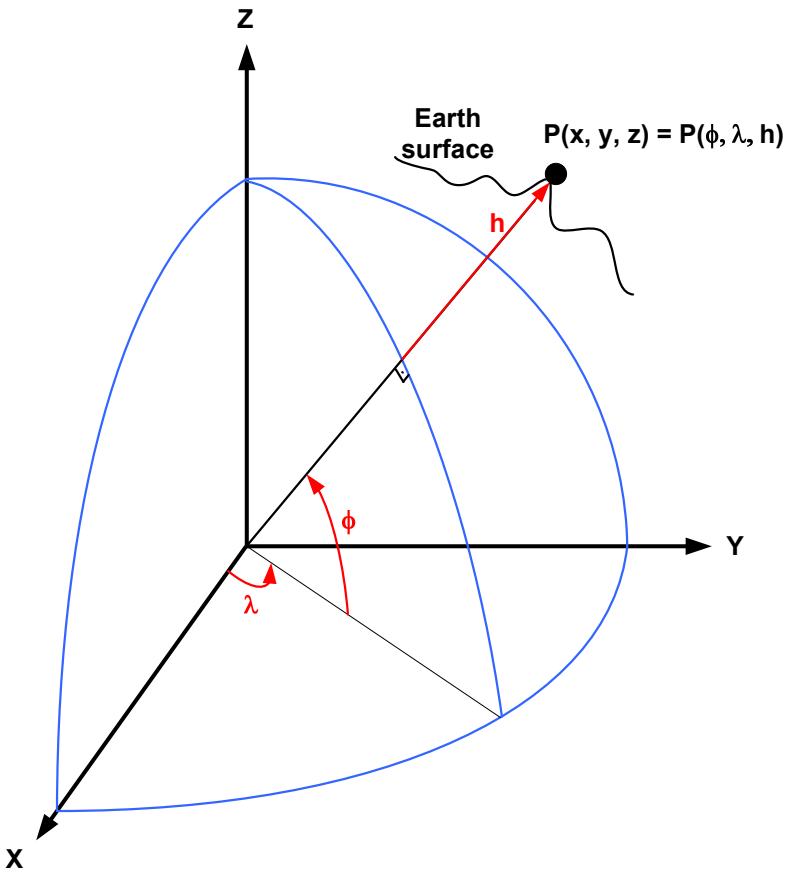


Figure 6.6: GPS position parameters in latitude (ϕ), longitude (λ) and altitude (h) and x, y and z in ECEF. Following [300].

nates $\mathbf{x_{gps}}$, $\mathbf{y_{gps}}$ and $\mathbf{z_{gps}}$ as:

$$\begin{aligned} x_{gps} &= (N+h)\cos(\phi)\cos(\lambda) \\ y_{gps} &= (N+h)\cos(\phi)\sin(\lambda) \\ z_{gps} &= ((1-e^2)N+h)\sin(\phi) \end{aligned} \tag{6.1}$$

where

$$N = \frac{a}{\sqrt{1.0 - e^2\sin(\phi)^2}} \tag{6.2}$$

and a is the WGS84 [309] ellipsoid constant for equatorial earth radius (6,378,137m), e^2 corresponds to the eccentricity of the earth with a value of $6.69437999 \times 10^{-3}$ [284]. The calculated values form the measurements from the GPS sensor as $m_{gps} = (x_{gps}, y_{gps}, z_{gps})$.

6.2.3 IMU motion estimate

The IMU is used for both calculating a position estimate that will be combined with estimates from other sensors and generating the orientation estimate using a recent IMU filter by Madgwick [6]. Before finding these motion estimates from this sensor it is important to find noise parameters using a simple calibration stage described in the following.

Sensor calibration

The Phidgets 1056 IMU used in the experiments is calibrated in the factory in order to prevent production-related problems such as sensor sensitivity and cross-axis misalignment. Nevertheless, the sensor generates non-zero values which are known as bias (offset) parameters at rest. Sensor calibration in this case simply consists of finding the values which are generated by the IMU while it is still.

It is performed by placing the IMU on a flat and stable surface (it was observed that even the vibrations from the computer can affect the parameters) and taking samples ($\sim$ 5000, which takes around 30 seconds). The samples for the accelerometer (a_x, a_y, a_z) and the gyroscope (g_x, g_y, g_z) are accumulated and their mean is found as the bias for each axis for both sensors. These offsets, presented in Table 6.4, are then subtracted from each reading to find the actual amount of acceleration or rate of turn.

Table 6.4: Calibration parameters found for accelerometer and gyroscope

	Offsets		
	x	**y**	**z**
Accelerometer	-0.000817	0.158242	0.987314
Gyroscope	-0.216527	-0.052387	-0.183611

In addition to finding the bias parameters and subtracting them from the readings, a second approach for reducing the noise is to accumulate a set of readings (*e.g.* four readings) and using their mean in order to reduce the effect of noise in position and orientation estimates from the IMU. Use of these calibration parameters and averaging several readings reduced the drift to a mean of 0.5° per minute when these parameters were used with the IMU filter described below.

Position and orientation estimates

Finding the position estimate from the IMU is performed by double-integrating the accelerometer outputs for several samples, the current implementation uses four samples. The first integration, to find the velocity, involves integrating ac-

celerations using $v(t) = v(0) + at$:

$$\begin{aligned} v_x &= \int_0^T a_x \mathrm{d}t = v_x(T) - v_x(0) \\ v_y &= \int_0^T a_y \mathrm{d}t = v_y(T) - v_y(0) \\ v_z &= \int_0^T a_z \mathrm{d}t = v_z(T) - v_z(0) \end{aligned} \tag{6.3}$$

Since multiple samples are taken, dt is the time passed for each one of them. The next step is to integrate the velocities from (6.3) to find the position using $x(t) = x(0) + vt$ as

$$\begin{aligned} x_{imu} &= \int_0^T v_x \mathrm{d}t = p_x(T) - p_x(0) \\ y_{imu} &= \int_0^T v_y \mathrm{d}t = p_y(T) - p_y(0) \\ z_{imu} &= \int_0^T v_z \mathrm{d}t = p_z(T) - p_z(0) \end{aligned} \tag{6.4}$$

These calculated positions ($m_{imu} = (x_{imu}, y_{imu}, z_{imu})$) are used as the measurements from the IMU, used in both combining estimates from other sensors for the fusion filter presented in this chapter and a conventional GPS–IMU sensor fusion developed for comparison–see section 6.5. For the values from accelerometer and gyroscope in Figures 6.7 and 6.8, the position estimate from the IMU can be seen in Figure 6.9 below.

The orientation estimate is calculated using the IMU filter due to [6] which can calculate orientation efficiently both for IMUs and MARGs. This filter uses the quaternion representation for avoiding problems related to the Euler representation such as *gimbal lock*, which occurs when the pitch is $\pm 90°$ in which case the heading and roll will rotate around the vertical axis, losing a DoF [310]. This problem is rather unlikely to occur for tracking system described here since the

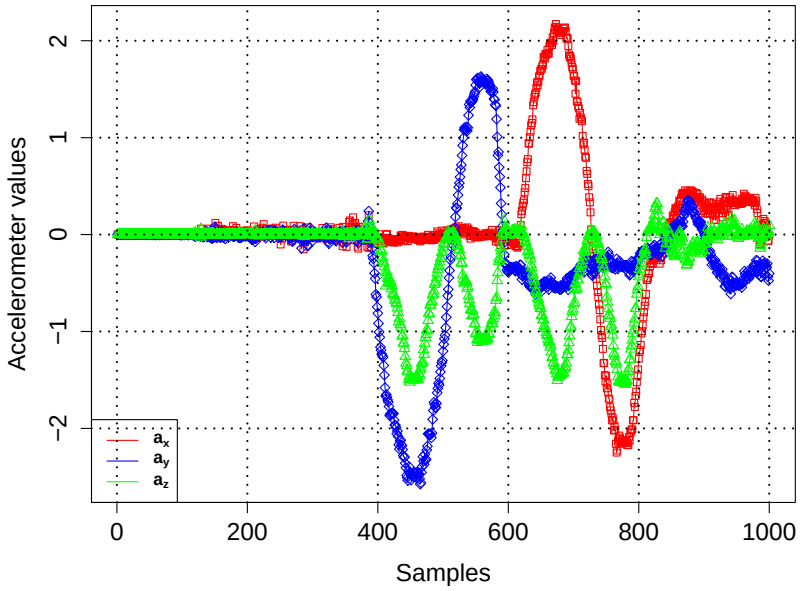


Figure 6.7: Sample values from the accelerometer

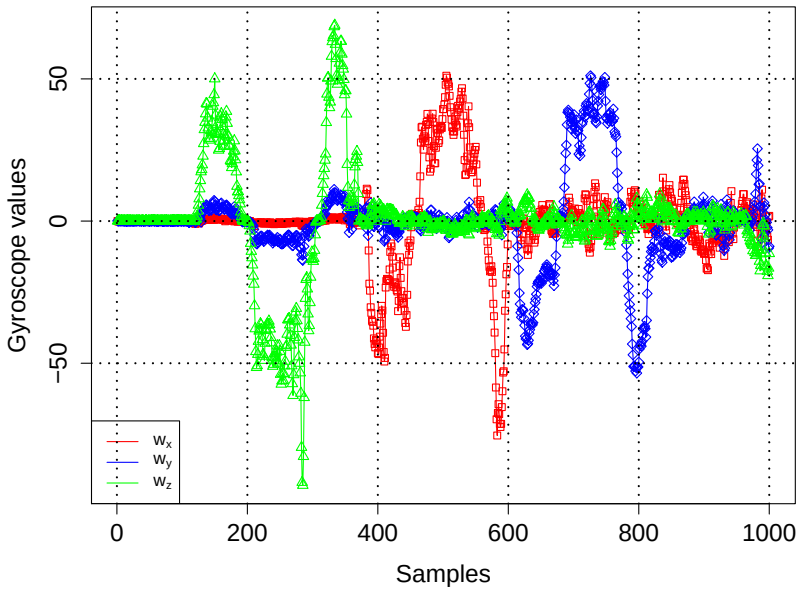


Figure 6.8: Sample values from the gyroscope

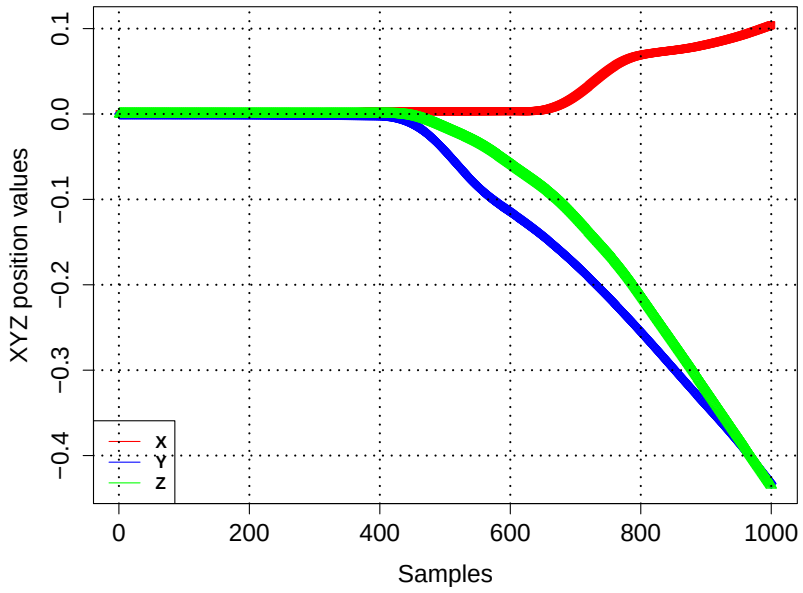


Figure 6.9: IMU estimates for the 3D position

user being tracked is not expected to look upwards but this feature of the Madgwick filter provides additional robustness. A gradient descent algorithm was used by this filter to obtain the final motion estimate by making good use of the high frequency outputs from inertial sensors [86] rather than performing several iterations.

Using the IMU filter in [6] described above the orientation is converted to Euler angles for the fusion filter described in section 6.3 and samples values are presented in Figure 6.10.

6.3 Sensor Fusion Algorithm

The motion estimates obtained by the individual sensors presented in section 6.2 are prone to error due to problems related to accuracy and drift. It makes sense

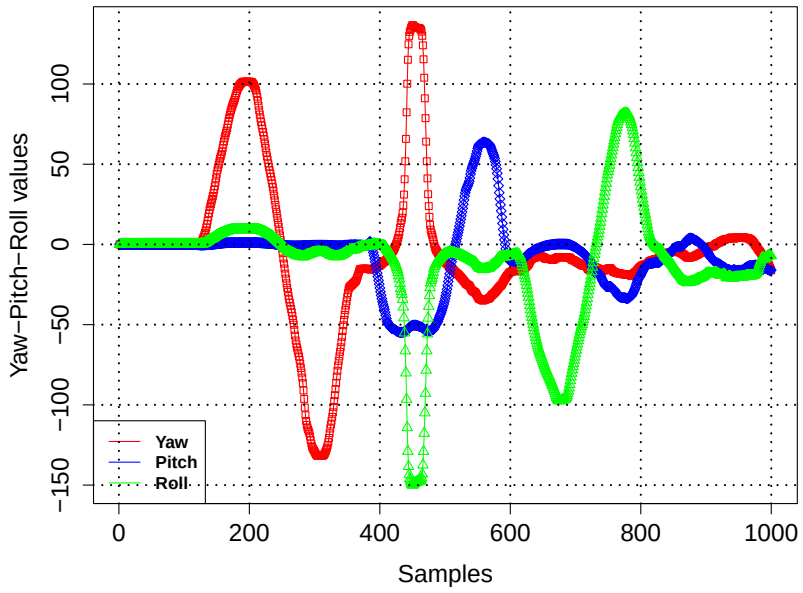


Figure 6.10: Yaw, pitch and roll values obtained using the filter of [6]

to combine measurements from several sensors in order to exploit their characteristics, which are complementary to each other (see section 2.7.1), in order to yield more accurate results. For this reason a sensor fusion approach was followed using a KF, since it is most common for such applications [284].

The following subsections elaborate on the fusion filter, describing how the motion estimates from the three sensors are combined in a tightly-coupled design, the approach making use of multiple threads for efficiency and finally the tracking system using the sensor fusion algorithm presented here.

6.3.1 Fusion filter

The filter designed for integration of three sensors consists of a state x which includes positional data ($P = (P_x, P_y, P_z)^T$), linear velocities ($V = (V_x, V_y, V_z)^T$),

rotational data ($R = (R_x, R_y, R_z)^T$) and angular velocities ($\Omega = (\Omega_x, \Omega_y, \Omega_z)^T$):

$$x = (P, V, R, \Omega)^T \tag{6.5}$$

A simple state consisting of 12 elements will facilitate obtaining a better performance in speed than one with a larger state. At each iteration, the predict–measure–update cycle of the KF is executed in order to produce a single output from several sensors as the filter output.

In the first stage, *i.e.* prediction, a transition matrix (F of (6.6)) is applied to the state x in order to obtain the predicted position:

$$F = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.6}$$

where Δt is the time between two prediction stages. This initial version of the transition matrix is relatively simple –using Constant Motion Model (CMM) which will be elaborated later in the chapter; however fuzzy rules, described in sec-

tion 6.4, will be used to decide on the velocity coefficients that will update this transition matrix.

The majority of the operations required for integrating the motion estimates from the three sensors are performed in the second stage, where measurements are taken and provided to the filter so that it can update itself. This stage can be examined separately for the position and orientation estimates. For the latter, the output of the IMU filter ($m_R = (yaw, pitch, roll)$ provide the rotational measurements used to update R in (6.5).

The idea of combining the positional estimates from the camera, GPS and IMU is due to the fact that the GPS is a discrete-time position sensor [284]. In order for the AR system used in applications presented in chapter 7 to update the position of the virtual camera more rapidly, the position estimates from the fusion filter need to have smooth transitions between them in order to provide the impression of continuous motion. This is achieved by applying the transformation Tr, obtained from the motion estimate of the camera, to the position provided by the GPS sensor (m_{gps}) and then adding the motion estimate of the IMU (m_{imu}) as an offset:

$$m_P = Tr \times m_{gps} + m_{imu} \tag{6.7}$$

where m_P constitutes the positional measurements for the fusion filter.

Having all the measurements (m_P and m_R) ready, the filter can now update itself using the last three lines of the KF (Algorithm 1). After the update, the obtained estimates can directly be used by the AR system described in chapter 7 in order to update the position of the virtual camera.

6.3.2 Multi-threaded approach

The sensors used in the system had different data rates for delivering data and performing calculations to produce a motion estimate. These frequency differences resulted in a challenge while combining them to generate a single output for the AR system. To elaborate, the vision-based system can produce up to 4 motion estimates per second while the IMU can produce up to 250 samples per second which are to be used for the orientation filter and GPS can produce only a single measurement every second. Furthermore, AR processing has to produce a minimum of 10fps in order to generate a smooth flow of the display.

A second challenge is due to the execution method of the GPS and the IMU sensors. The library handling these sensors was designed to be event-driven (*i.e.* an event is triggered each time a datum is available.). The library makes an automatic call the related event handler when data from any of these two sensors become ready. This did not allow the handling of these sensors in the same thread, where other computations are performed in a procedural manner.

Due to the differences in data rates and the application logic used for the sensors employed in the system, a multi-threaded approach was followed as shown in Figure 6.11. The design used two child threads in addition to the main thread in order to circumvent the challenges mentioned above. The main thread is used to acquire camera images and pass these to both of the vision-based method as keyframes and AR processing. A child thread is used to handle vision-based processing by accessing the camera images acquired. The algorithm generates an estimate which will be later used by the fusion filter in the main thread. A second child thread handles the GPS and IMU sensors and computes the estimate generated by them.

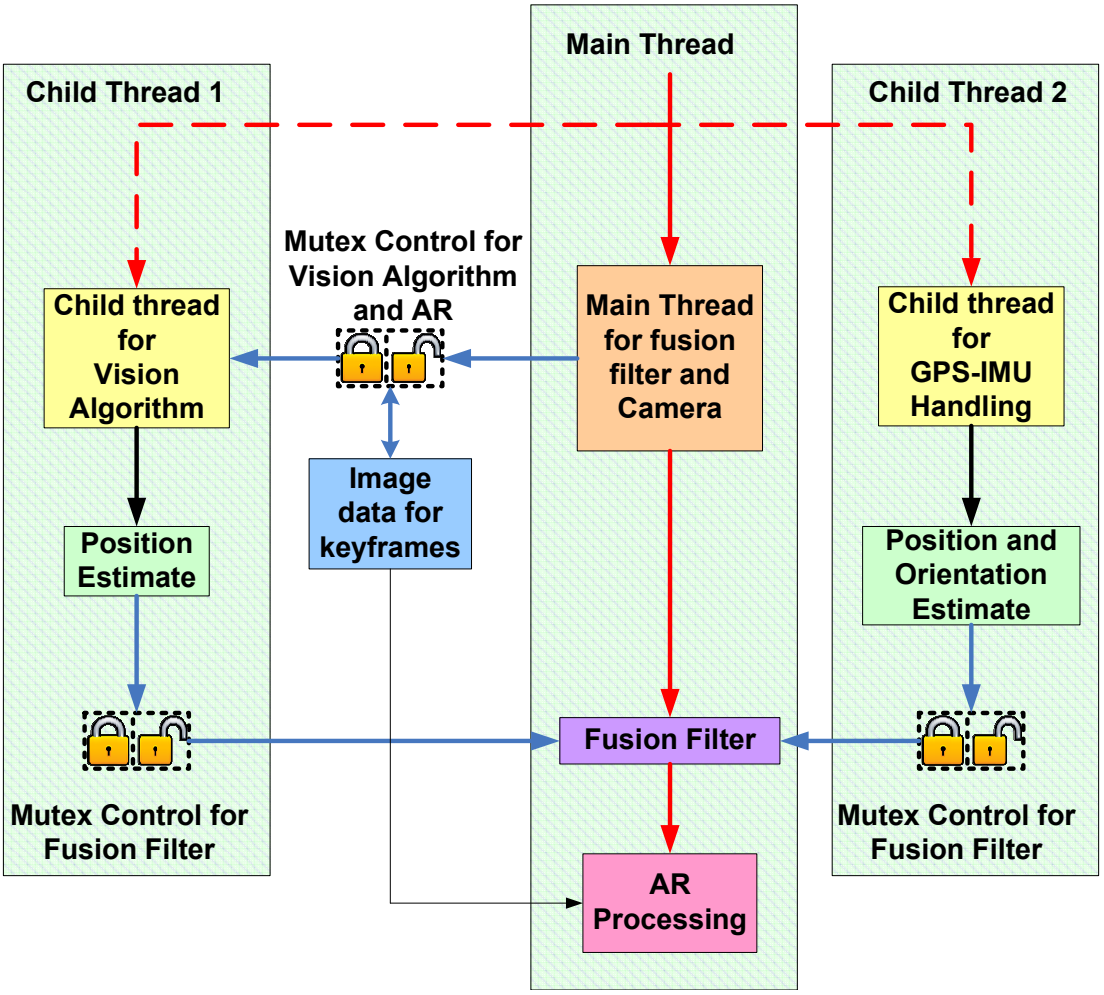


Figure 6.11: Diagram of using multiple threads to access data from different sensors. The main thread is responsible for acquiring camera images, the fusion filter and using these two generating the AR output. The GPS and IMU handler is handled by a child thread since it is working based on events generated by these two sensors. The second child thread is responsible for the vision algorithm. Race conditions are prevented using three MutExes (shown with locks).

When the child threads have an estimate, they are joined at the fusion filter running in the main thread. The fusion filter produces a single final estimate of position and orientation which will be used as the viewpoint parameters for AR processing along with the image acquired by the camera.

It is also worth mentioning that the design here followed a producer-consumer approach. The child threads are mainly responsible for producing the position/orientation estimate, while the fusion filter consumes these estimates. In order to prevent corruption of data in shared locations (*e.g.* the camera image or values storing the estimates from vision-based method and other sensors), MutExes [311] were used to prevent cases where the two threads might be competing to access shared information at the same time (*i.e.* where race conditions occur).

The approach described here may appear as an implementation detail, but is essential if one is to show that the system is working based on the principles laid out in section 2.9 (*i.e.* using parallelism for performance). This approach resulted in satisfactory frame rates in the final application.

6.3.3 Tracking system

The fusion algorithm was tested on a simple tracking system designed for this study. As shown in Figure 6.12, the system consists of a laptop computer (Intel, dual core 2.80Ghz, 4GB memory with Linux operating system), a GPS receiver, an IMU and a web camera. An external power supply was also required for the hub connecting the sensors to the laptop since a single port was not able to provide enough power for the three sensors.

The placement of the sensors was chosen considering the lever-arm effect [312] which occurs particularly in GPS–IMU integration systems when sensors are

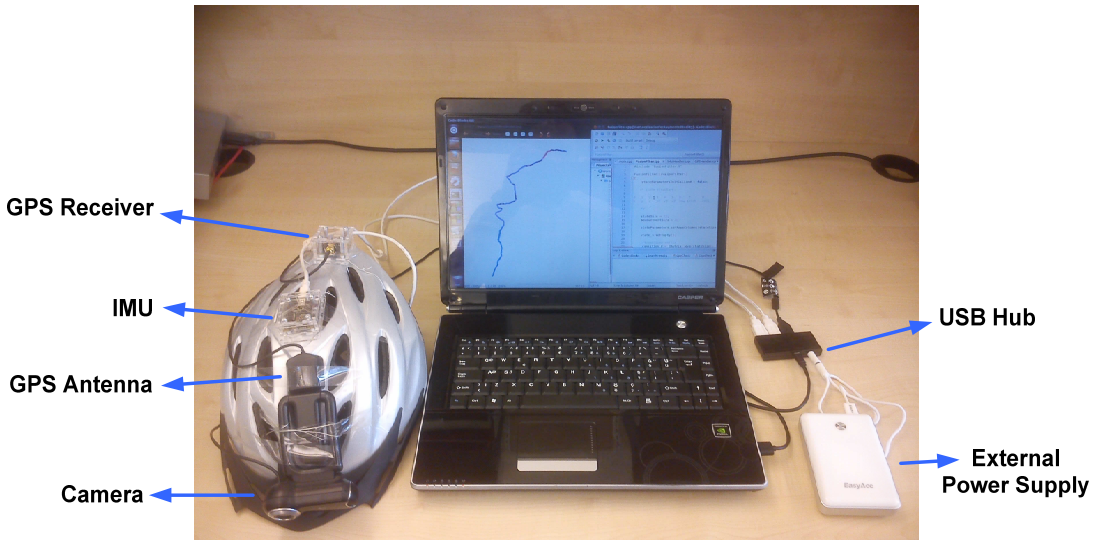


Figure 6.12: Tracking system

placed apart from each other, so that the positional and rotational data sensed by them correspond to different positions. With this in mind, the camera, IMU and the antenna of the GPS are all placed 2–3 cm apart from each other, a negligible distance.

6.4 Fuzzy Logic Based Multiple Motion Models

There are a number of uncertainties related to the sensors used for obtaining motion estimates as measurements for the fusion filter. These uncertainties and imprecisions arise from the accuracy problems of the GPS, loss of fine motion detail in the vision-based approach and drift problems in case of the IMU. A fusion of multiple sensors and combining estimates from them significantly reduces these problems and provide more accurate results.

Another reason that causes tracking problems is actually a different source of uncertainty, the motion patterns followed by the user. A user, in a cultural

heritage context, may follow a number of motion patterns, which may include stopping to examine ruins, walking slowly looking around or walking with a higher speed. The filter developed in section 6.3.1 uses a constant transition function (F in (6.6)) which does not take into account any information about the actual motion pattern performed by the user in the prediction stage. An improvement can be achieved if the filter is dynamically adapted based on the user's motion patterns. The reason behind this is that the dynamics of the filtering process is governed by both the internal parameters of the filter, such as the state x and noise parameters (Q for the process and R for the measurement noise, see Figure 4.5), and on the external side by the motion model used in F for prediction.

This section presents Fuzzy Adaptive Motion Model (FAMM), a method of employing fuzzy logic to decide which one of the several motion models is the best fitting one, so that the filter state will be more in line with the measurements and hence converge faster.

6.4.1 Handling the uncertainty in fusion filter

The idea presented here is to use adaptive motion models depending on the KF innovation (y) with an attempt to minimize the filter error. The innovation is actually hidden in the second line of the update part in the KF in Algorithm 1 where the filter is updated in order to obtain the next value of the state using the measurements:

$$x_{i+1} = \hat{x}_{i+1} + K_i(z_i - h_i\hat{x}_{i+1}) \tag{6.8}$$

The difference between the measurements (z) and the prediction ($h\hat{x}$), omitting

the subscripts indicating time, is defined as the innovation (y):

$$y = z - h\hat{x} \tag{6.9}$$

For the filter designed in the previous section, the innovation y can be broken into two parts for positional (y_p) and rotational (y_r) data by taking the corresponding matrix elements. These elements refer to the measurement errors for positional and rotational estimates and will be used to decide on the motion model to be used in the next prediction stage (Figure 6.13) with the aim of reducing filter error.

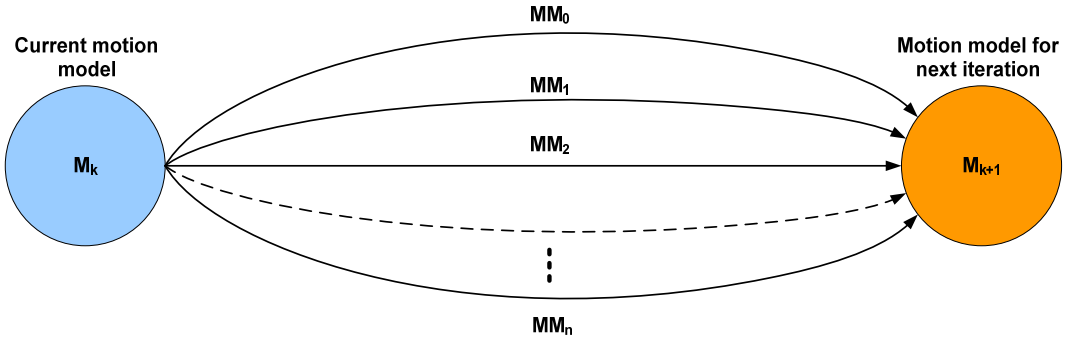


Figure 6.13: Selection of the motion model for next prediction stage.

The design here makes use of nine motion models MM, each denoted as $\mathtt{P}i\mathtt{R}j$ where $i, j \in (0, 1, 2)$. Values for i and j are considered as velocity coefficients (c_i and c_j) for the two components of the transition function (F) for position

$$\hat{x}_P = x_P + c_i V \Delta t \tag{6.10}$$

and orientation as

$$\hat{x}_R = x_R + c_j \Omega \Delta t \tag{6.11}$$

The idea presented here can be best described using examples of how these motion models work. For instance, `P0R0` indicates a stationary transition model where the current values of the state (6.5) for position (P) and rotation (R) will be unchanged in the predicted state, whereas `P1R2` indicates a motion model where position is predicted with current positional velocities ($\hat{x}_P = x_P + (1V)\Delta t$) but rotational velocities are doubled ($j = 2$ so $\hat{x}_R = x_R + (2\Omega)\Delta t$) to compensate for the effects of severe rotations.

Actual selection of the motion model is achieved using a FLC which takes the y_p and y_r parameters, calculated in the update stage of the filter. In the implementation, magnitudes of y_p and y_r are calculated and then the input membership functions are applied to them. The output of the membership function will define the 'firing strengths' of rules. The rule with the maximum firing strength is selected to choose the motion model that will be used in the next prediction stage. This process is illustrated in Figure 6.14, is essentially a more detailed exposition of Figure 6.13.

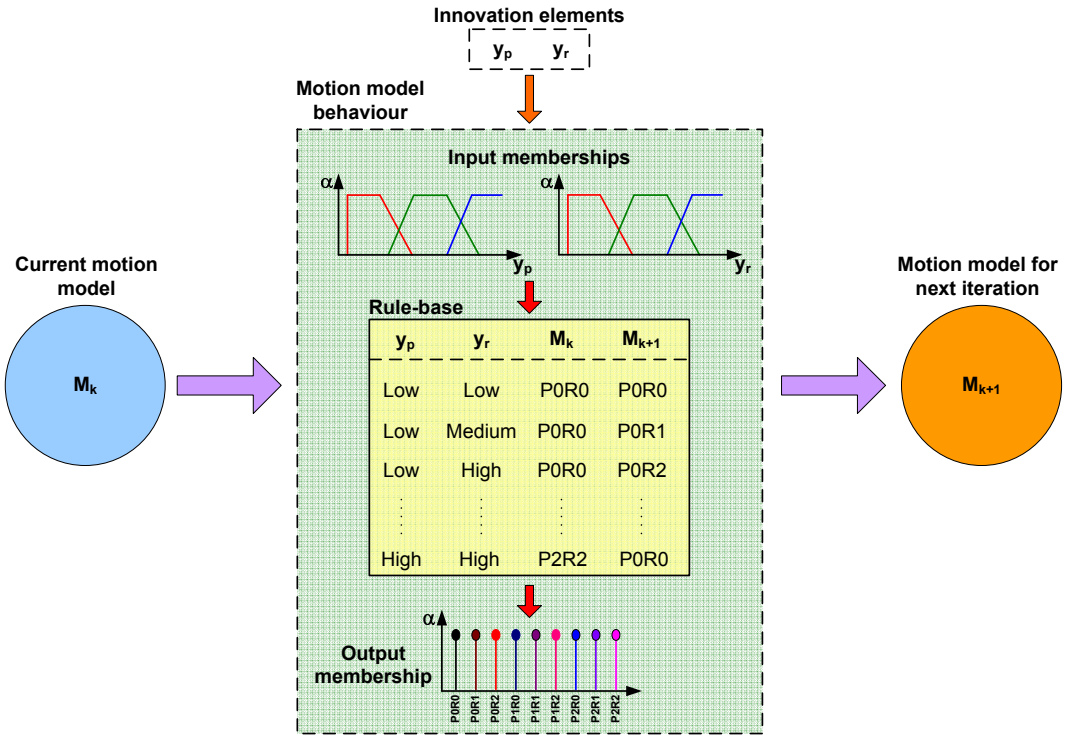


Figure 6.14: Fuzzy rule-based selection of motion models. The FLC takes the two components of the innovation (*i.e.* positional and rotational) and applies the membership functions in order to decide the firing strengths of the rules available in the rule-base. The antecedents of the rules are used to define the motion model used in the prediction stage of the next filter iteration.

Using the innovation values, a rule-base was designed for motion models which are used to fill the components of the transition matrix (F) to provide this functionality.

6.4.2 Rule-base definition

The rule-base consists of the rules which can be examined in two parts namely the antecedent and the consequent. The antecedent part defines the conditions to be satisfied for the consequent to occur. In this case, the antecedents will

include the membership values for the positional and rotational innovations and the current type of the motion model in order to select the motion model for the next prediction as the consequent.

A rule of the form `<Low, Medium, P0R0, P0R1>`, uses the first three components as antecedents and the last as the consequent, should be read as:

> "*IF* the positional innovation is `Low` *AND*
> rotational innovation is `Medium` *AND*
> the current motion model (M_k) is `P0R0`, *THEN*
> change the motion model to `P0R1` (M_{k+1}) for the next iteration of the filter."

The rule-base presented in Table 6.5 consists of $3^4 = 81$ rules l^n where l is the number of linguistic variables (three for `(Low, Medium,High)` and n is the number of input variables (four for y_p, y_r and M_k which counts for two variables since $M_k = P_i R_j$).

Table 6.5: Rule-base for multiple motion models

Position	**Rotation**	**Current Model (M_k)**	**Selected Model (M_{k+1})**
Low	Low	P0R0	P0R0
Low	Medium	P0R0	P0R1
Low	High	P0R0	P0R2
Low	Low	P0R1	P0R1
Low	Medium	P0R1	P0R2
Low	High	P0R1	P0R0
Low	Low	P0R2	P0R2

Continued on next page

Table 6.5 – *Continued from previous page*

Position (y_p)	**Rotation (y_r)**	**Current Model**	**Selected Model**
Low	Medium	P0R2	P0R1
Low	High	P0R2	P0R0
Low	Low	P1R0	P1R0
Low	Medium	P1R0	P1R1
Low	High	P1R0	P1R2
Low	Low	P1R1	P1R1
Low	Medium	P1R1	P1R2
Low	High	P1R1	P1R0
Low	Low	P1R2	P1R2
Low	Medium	P1R2	P1R1
Low	High	P1R2	P1R0
Low	Low	P2R0	P2R0
Low	Medium	P2R0	P2R1
Low	High	P2R0	P2R2
Low	Low	P2R1	P2R1
Low	Medium	P2R1	P2R2
Low	High	P2R1	P2R0
Low	Low	P2R2	P2R2
Low	Medium	P2R2	P2R1
Low	High	P2R2	P2R0
Medium	Low	P0R0	P1R0
Medium	Medium	P0R0	P1R1
Medium	High	P0R0	P1R2

Continued on next page

Table 6.5 – *Continued from previous page*

Position (y_p)	**Rotation (y_r)**	**Current Model**	**Selected Model**
Medium	Low	P0R1	P1R1
Medium	Medium	P0R1	P1R2
Medium	High	P0R1	P1R0
Medium	Low	P0R2	P1R2
Medium	Medium	P0R2	P1R1
Medium	High	P0R2	P1R0
Medium	Low	P1R0	P2R0
Medium	Medium	P1R0	P2R1
Medium	High	P1R0	P2R2
Medium	Low	P1R1	P2R1
Medium	Medium	P1R1	P2R2
Medium	High	P1R1	P2R0
Medium	Low	P1R2	P2R2
Medium	Medium	P1R2	P2R1
Medium	High	P1R2	P2R0
Medium	Low	P2R0	P1R0
Medium	Medium	P2R0	P1R1
Medium	High	P2R0	P1R2
Medium	Low	P2R1	P1R1
Medium	Medium	P2R1	P1R2
Medium	High	P2R1	P1R0
Medium	Low	P2R2	P1R2
Medium	Medium	P2R2	P1R1

Continued on next page

Table 6.5 – *Continued from previous page*

Position (y_p)	**Rotation (y_r)**	**Current Model**	**Selected Model**
Medium	High	P2R2	P1R0
High	Low	P0R0	P2R0
High	Medium	P0R0	P2R1
High	High	P0R0	P2R2
High	Low	P0R1	P2R1
High	Medium	P0R1	P2R2
High	High	P0R1	P2R0
High	Low	P0R2	P2R2
High	Medium	P0R2	P2R1
High	High	P0R2	P2R0
High	Low	P1R0	P0R0
High	Medium	P1R0	P0R1
High	High	P1R0	P0R2
High	Low	P1R1	P0R1
High	Medium	P1R1	P0R2
High	High	P1R1	P0R0
High	Low	P1R2	P0R2
High	Medium	P1R2	P0R1
High	High	P1R2	P0R0
High	Low	P2R0	P0R0
High	Medium	P2R0	P0R1
High	High	P2R0	P0R2
High	Low	P2R1	P0R1

Continued on next page

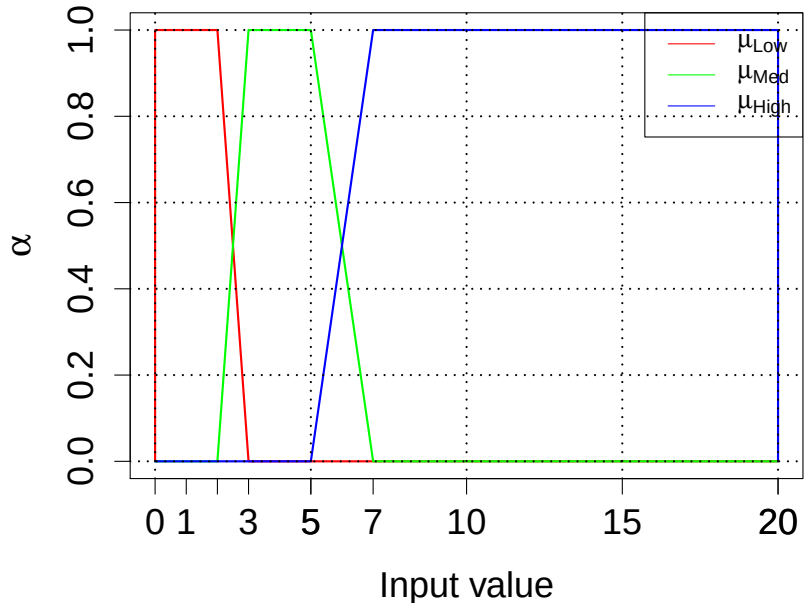
Table 6.5 – *Continued from previous page*

Position (y_p)	**Rotation (y_r)**	**Current Model**	**Selected Model**
High	Medium	P2R1	P0R2
High	High	P2R1	P0R0
High	Low	P2R2	P0R2
High	Medium	P2R2	P0R1
High	High	P2R2	P0R0

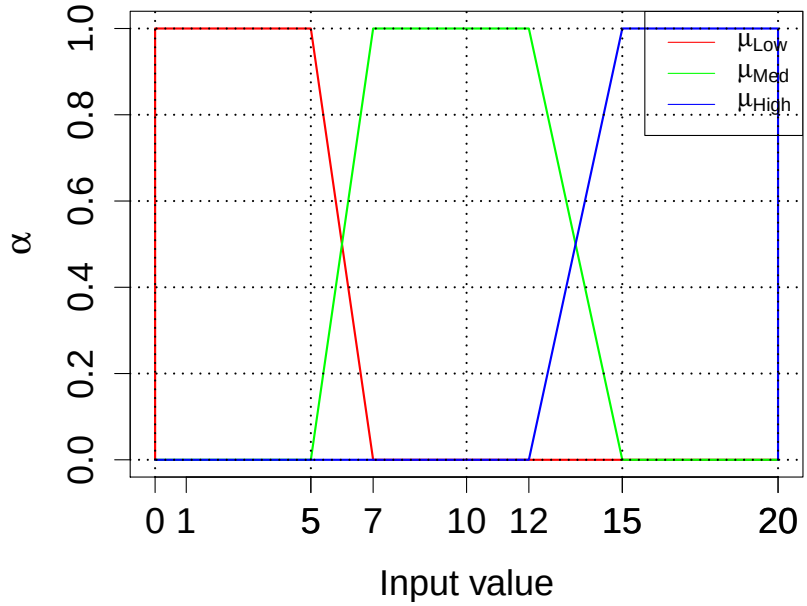
The rule-base presented here consists of a relatively large number of rules in order to handle all different transitions between motion models. For this reason, the rules are stored in a look-up table so that they can be accessed with a single query on the antecedent parameters. This design did not bring an extra computational overhead to the system, which is already using current resources at optimal capacity.

6.4.3 Input/Output membership functions

Earlier, it was mentioned that different velocity coefficients were used to allow a multiple motion models. These coefficients ($i, j \in (0, 1, 2)$) correspond to three fuzzy sets corresponding to three linguistic variables: `Low`, `Medium` and `High`. Calculation of the membership degrees for these linguistic variables are performed using the input membership functions defined as in Figure 6.15 using the crisp values of y_p and y_r.



(a) Positional innovation



(b) Rotational innovation

Figure 6.15: Input membership functions for positional and rotational innovations

The output membership function of the FLC is a singleton suggesting only one type of motion model based on the results of the input membership function and the rule-base, as shown in Figure 6.16. Note that the colours used to describe motion models will be used indicate the type of the motion model employed for different sections of the trajectory in section 6.5.

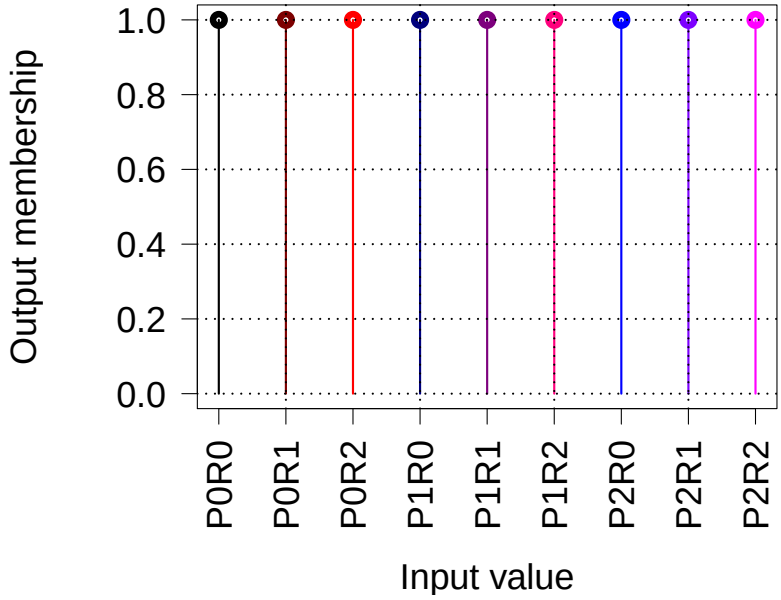


Figure 6.16: Output membership function

6.4.4 Processing

The FLC is implemented as a motion model behaviour which receives the positional and rotational innovations are parameters. The *AND* logical connector is represented using the product `t-norm` [126]. The firing strength of each is calculated by multiplying the membership values (μ_{Low}, μ_{Med} and μ_{High}) of positional and rotational innovations. The fire strengths for the rules which do not include the current motion model in its third antecedent are simply set to zero and the

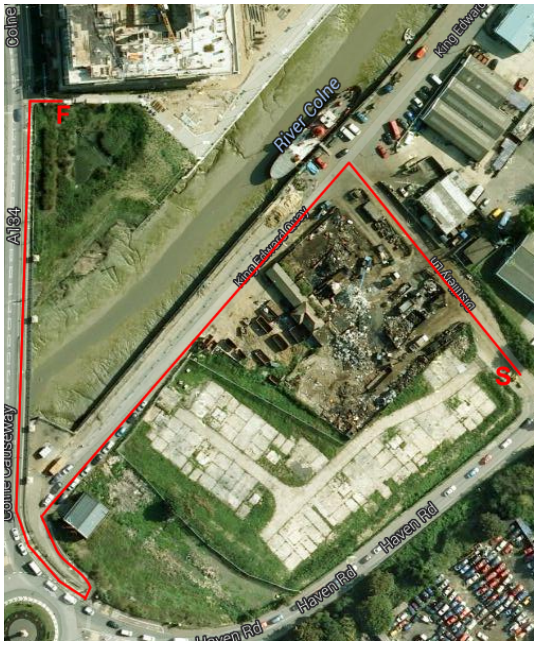
consequent of the rule with the maximum fire strength is selected as the motion model for the next prediction step.

6.5 Results

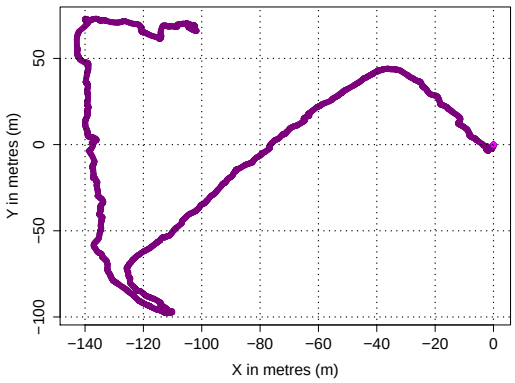
Figures 6.17 to 6.21 show the ground truth path and the estimated paths using integration of different sensors and employing different motion models based on the FAMM presented in section 6.4. Portions of the estimated paths are coloured differently, emphasizing the type of the motion model used for estimation. It is important to note that the CMM used in the figures correspond to `P1R1` and hence is drawn in the same colour.

Figure 6.19 shows data when the tracking system was completely stationary (*i.e.* no positional or rotational motion). Note how the motion is correctly estimated in (d) and (e) as `P0R0` which corresponds to a stationary motion model. A second thing to mention here is that the positional accuracy has been reduced to $\simeq$ 1 metres when the GPS is used with other sensors – an improvement on the results in Figure 6.4, where the positional accuracy was found as more than 2.5 metres.

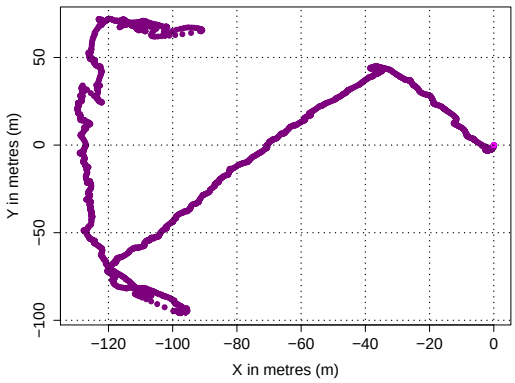
The advantage of using a camera and the FAMM is clearly shown in Figure 6.21, an example case for loop–closing. The integration of GPS and IMU could not handle the last segment of the path both when CMM (b) and FAMM (d) are used. In (c), the last segment was identified using integration of the camera with the two other sensors; however, the direction was not correct. Employing FAMM with the three sensors, shown in (e), gives the most accurate estimation. Furthermore, the overall shape of the estimated trajectory is closest to the ground truth path.



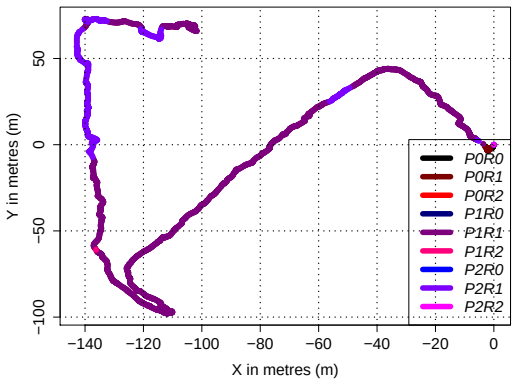
(a) Ground truth data



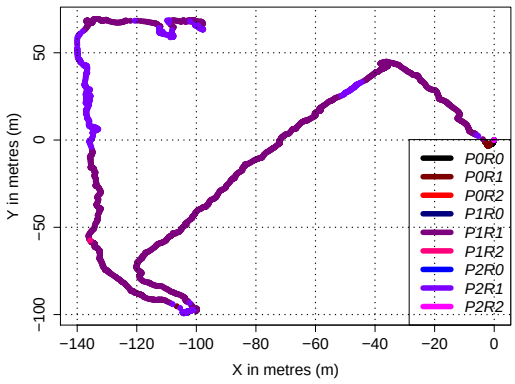
(b) Path using GPS and IMU with CMM



(c) Path using GPS, camera and IMU with CMM



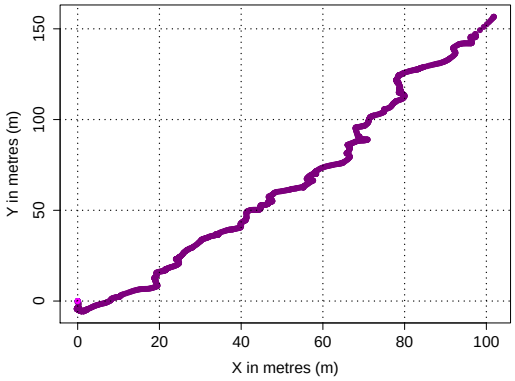
(d) Path using GPS and IMU with FAMM



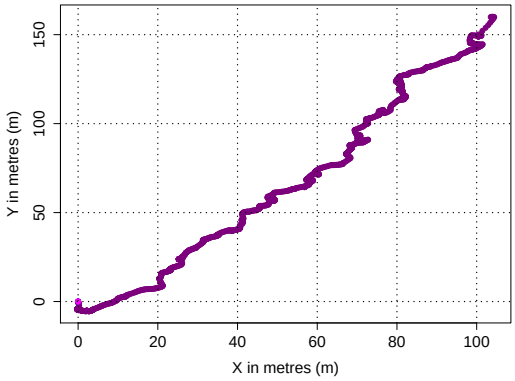
(e) Path using GPS, camera and IMU with FAMM

Figure 6.17: Real and estimated paths for dataset 1. Colours indicate the type of the motion model employed for estimating a part of the path.

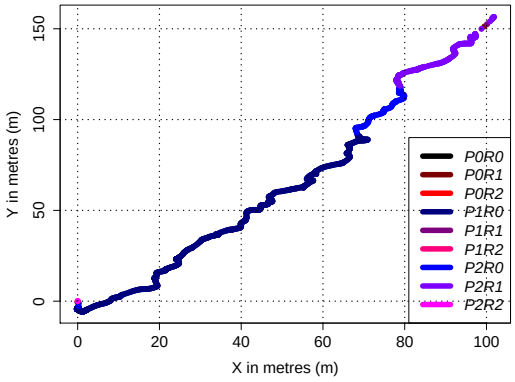
(a) Ground truth data



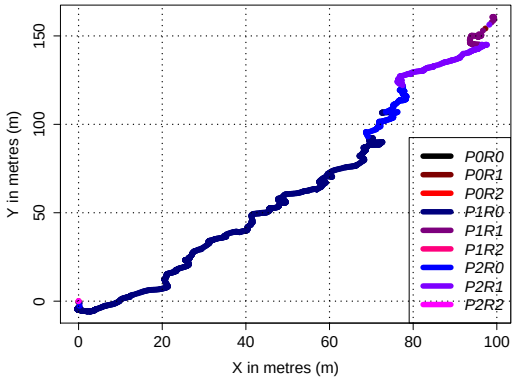
(b) Path using GPS and IMU with CMM



(c) Path using GPS, camera and IMU with CMM



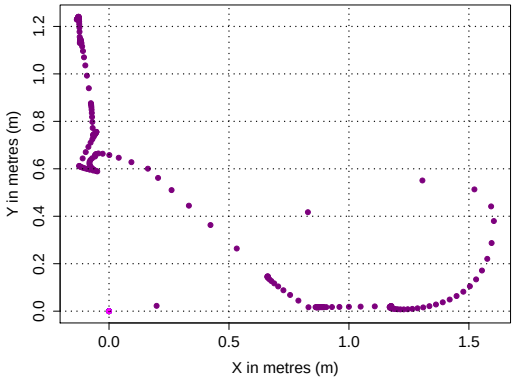
(d) Path using GPS and IMU with FAMM



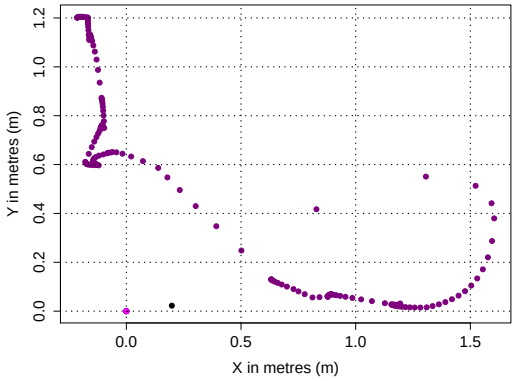
(e) Path using GPS, camera and IMU with FAMM

Figure 6.18: Real and estimated paths for dataset 2. Colours indicate the type of the motion model employed for estimating a part of the path.

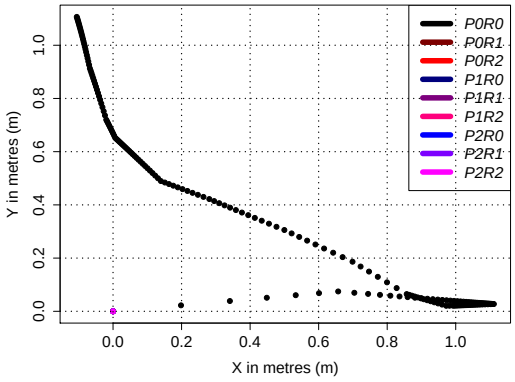
(a) Ground truth data



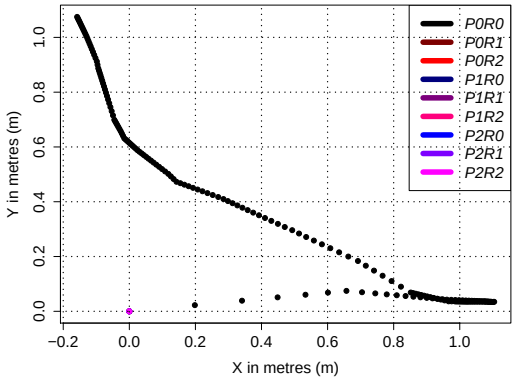
(b) Path using GPS and IMU with CMM



(c) Path using GPS, camera and IMU with CMM

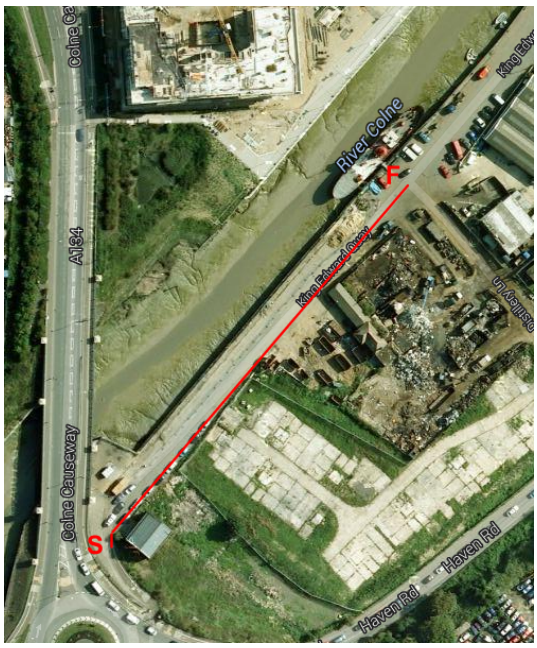


(d) Path using GPS and IMU with FAMM

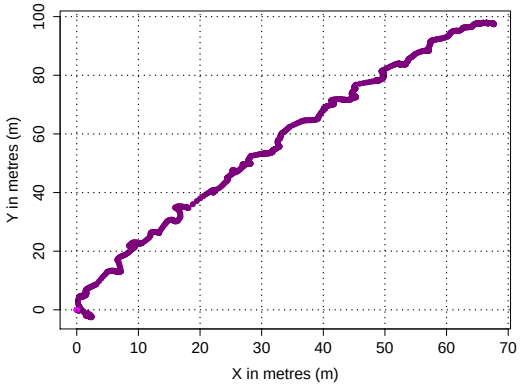


(e) Path using GPS, camera and IMU with FAMM

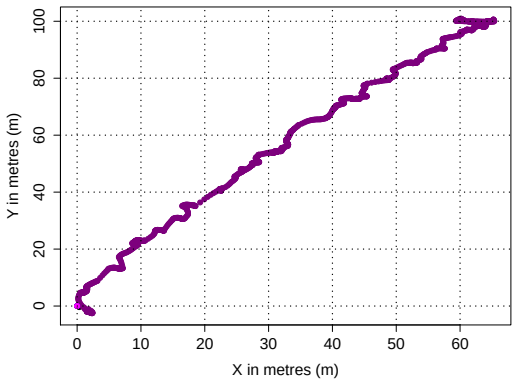
Figure 6.19: Real and estimated paths for dataset 3. Colours indicate the type of the motion model employed for estimating a part of the path.



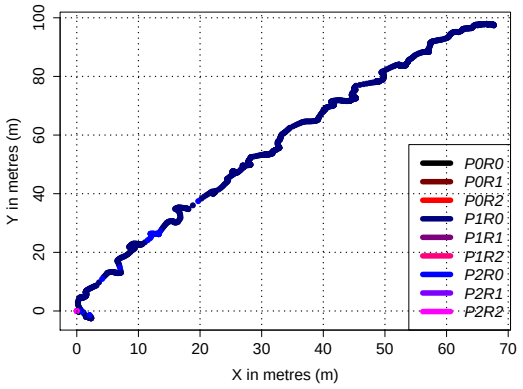
(a) Ground truth data



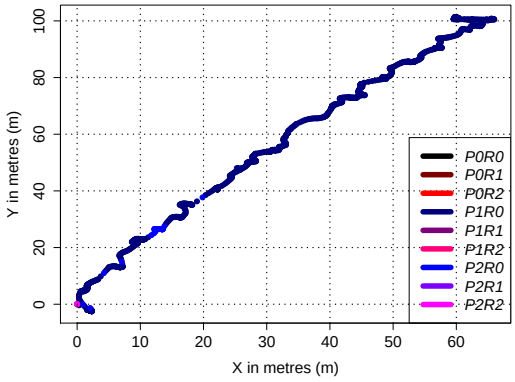
(b) Path using GPS and IMU with CMM



(c) Path using GPS, camera and IMU with CMM



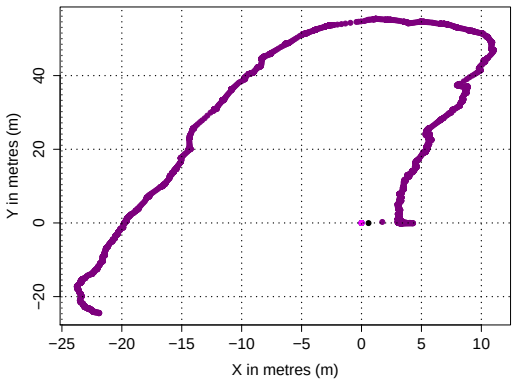
(d) Path using GPS and IMU with FAMM



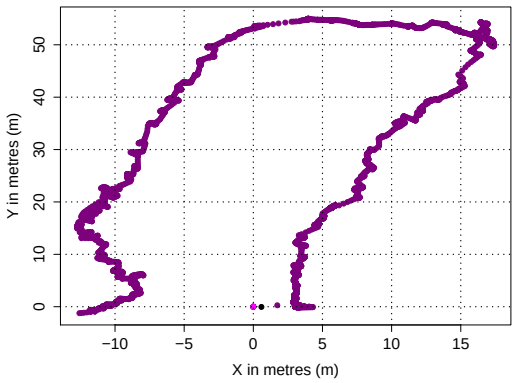
(e) Path using GPS, camera and IMU with FAMM

Figure 6.20: Real and estimated paths for dataset 4. Colours indicate the type of the motion model employed for estimating a part of the path.

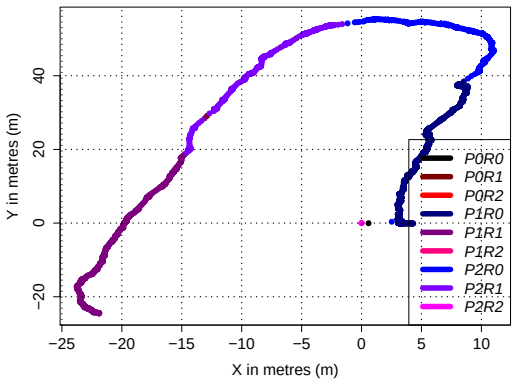
(a) Ground truth data



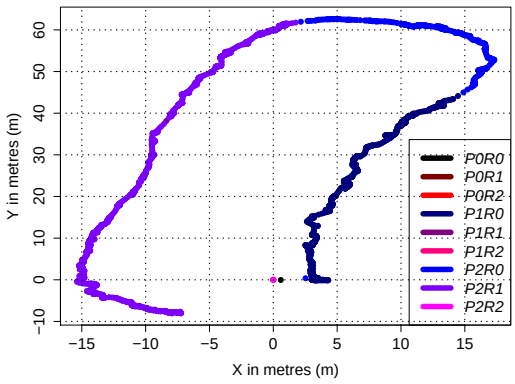
(b) Path using GPS and IMU with CMM



(c) Path using GPS, camera and IMU with CMM



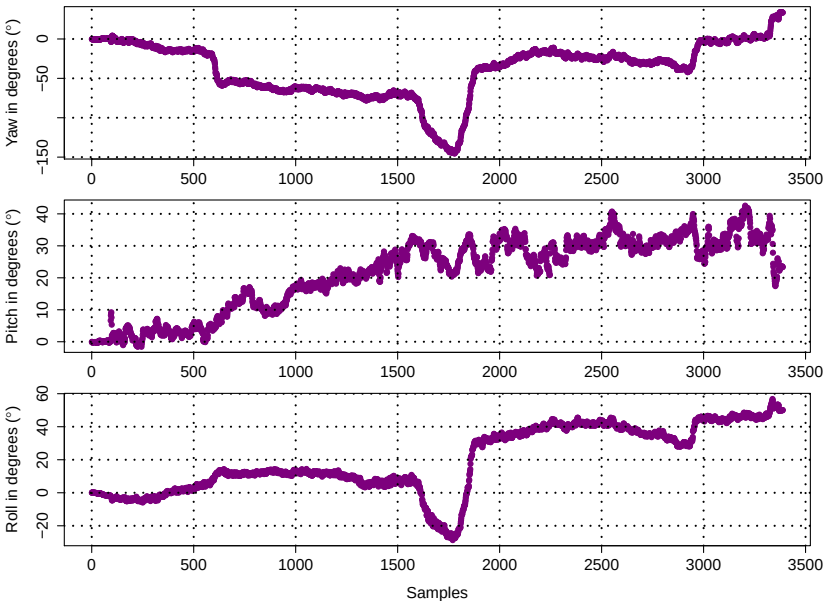
(d) Path using GPS and IMU with FAMM



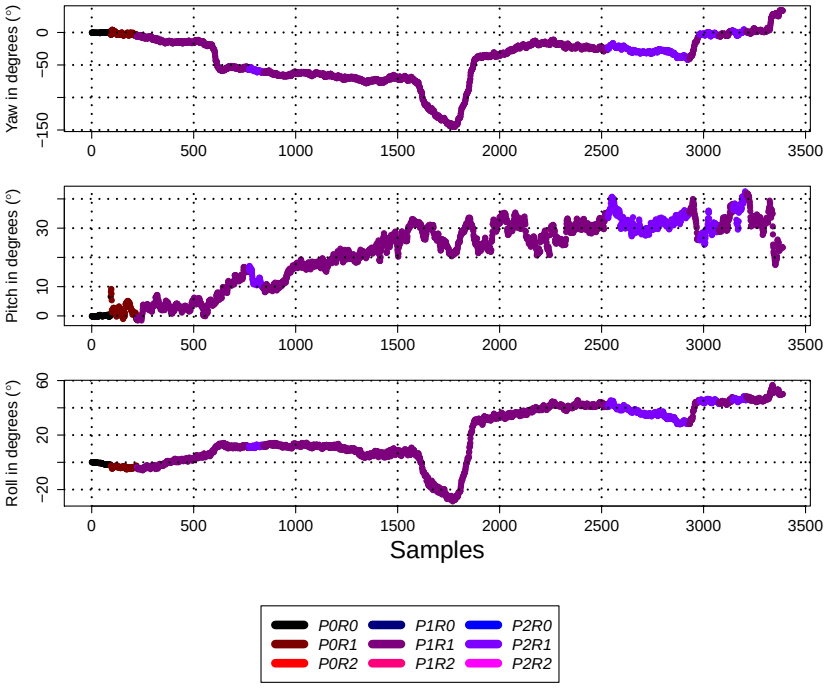
(e) Path using GPS, camera and IMU with FAMM

Figure 6.21: Real and estimated paths for dataset 5. Colours indicate the type of the motion model employed for estimating a part of the path.

Figures 6.22 to 6.23 present the estimated orientations for the datasets using the CMM and FAMM. One thing to mention in these orientation plots is that there is less jitter when the FAMM is employed for the motion model.

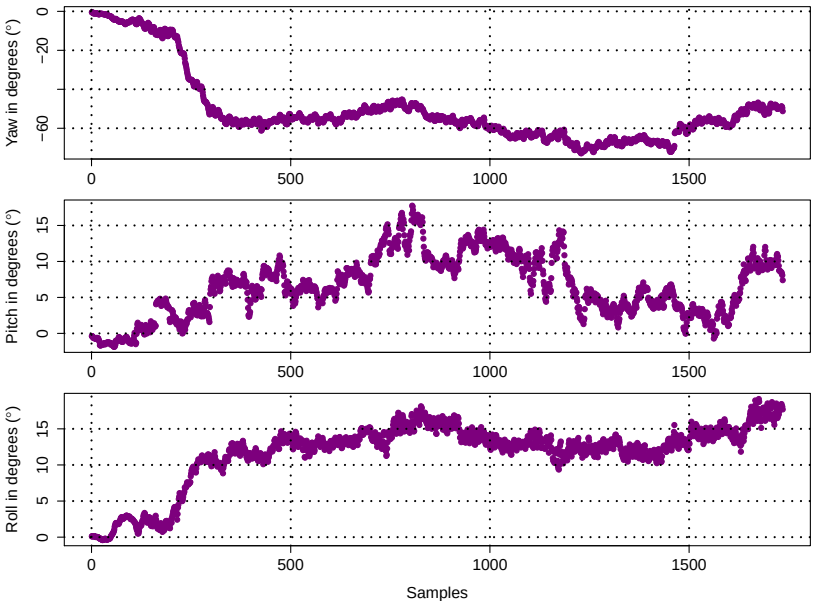


(a) Orientation using CMM

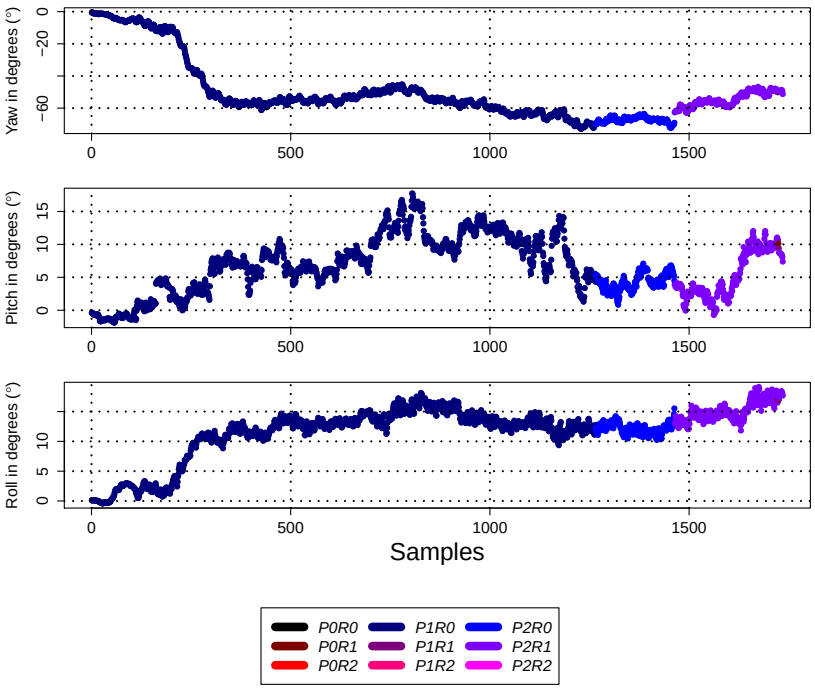


(b) Orientation using FAMM

Figure 6.22: Estimated rotations for CMM and FAMM for dataset 1. Colours indicate the type of the motion model used to estimate the orientation.

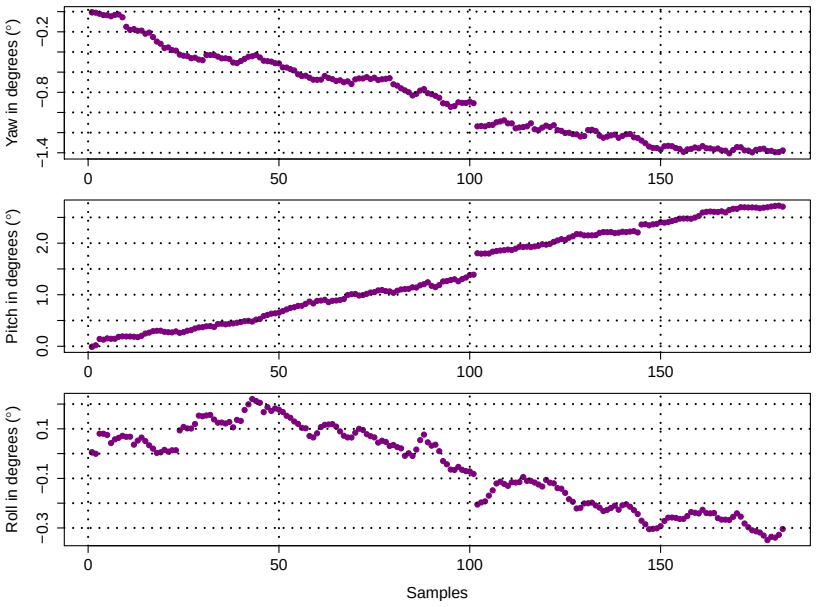


(a) Orientation using CMM

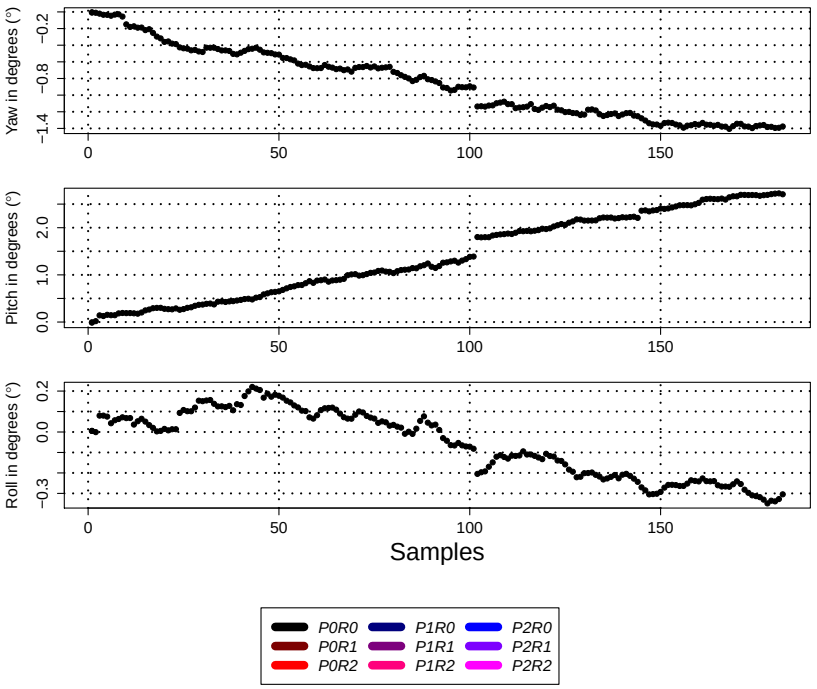


(b) Orientation using FAMM

Figure 6.23: Estimated rotations for CMM and FAMM for dataset 2. Colours indicate the type of the motion model used to estimate the orientation.

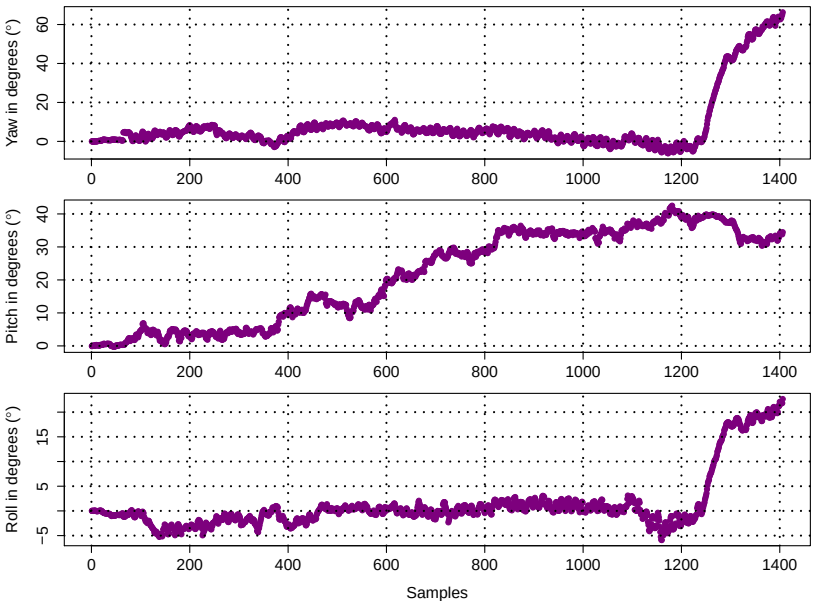


(a) Orientation using CMM

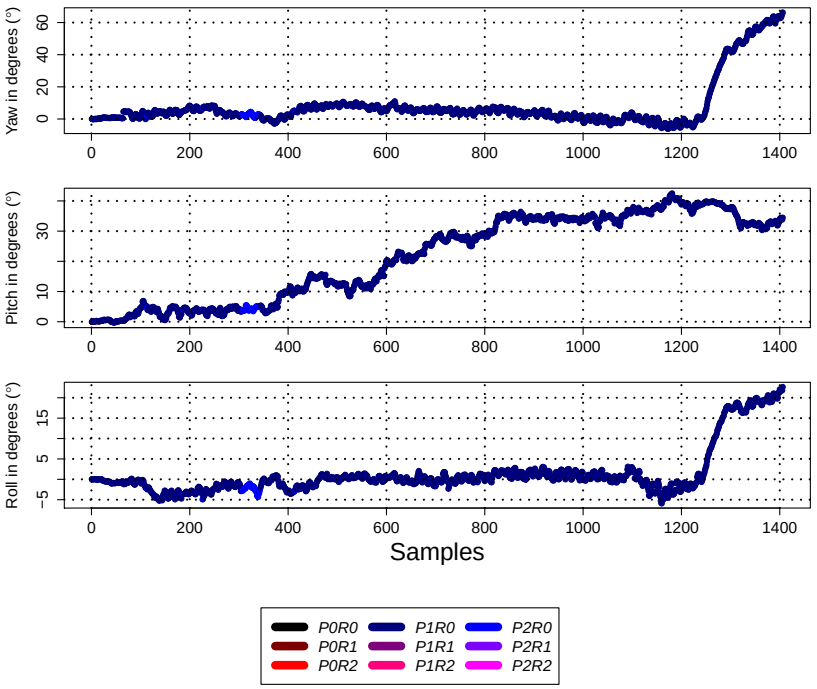


(b) Orientation using FAMM

Figure 6.24: Estimated rotations for CMM and FAMM for dataset 3. Colours indicate the type of the motion model used to estimate the orientation. Note the discontinuities in estimated rotations, which can be around 1°.

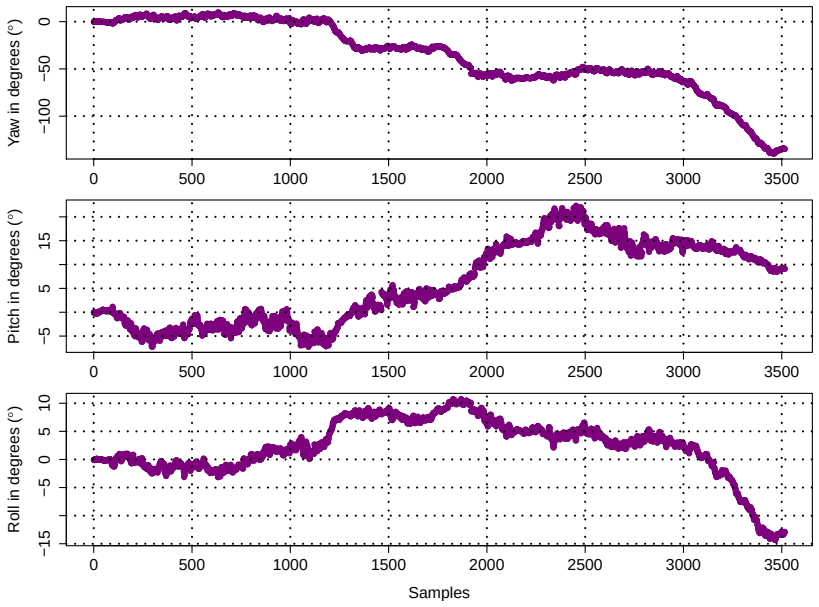


(a) Orientation using CMM

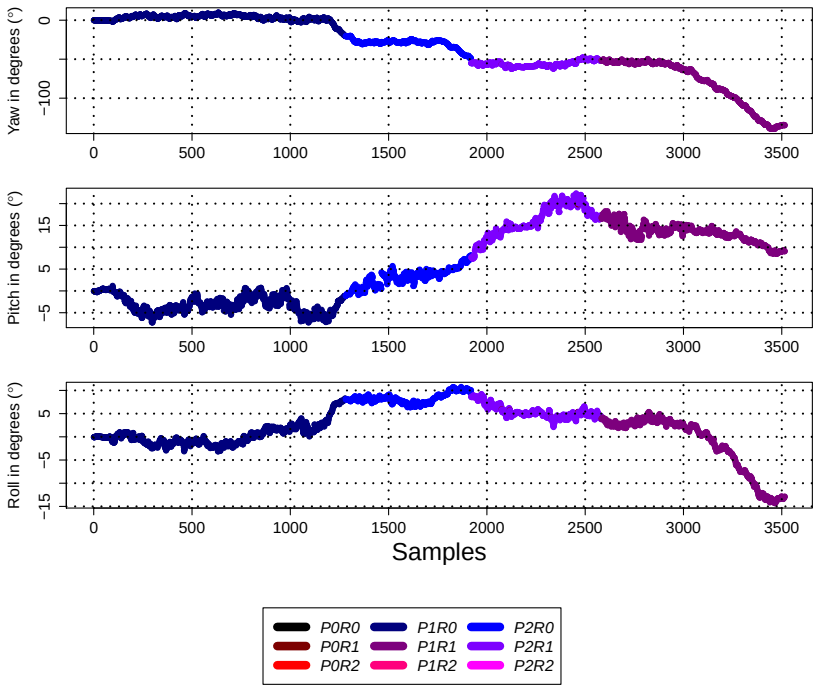


(b) Orientation using FAMM

Figure 6.25: Estimated rotations for CMM and FAMM for dataset 4. Colours indicate the type of the motion model used to estimate the orientation.



(a) Orientation using CMM

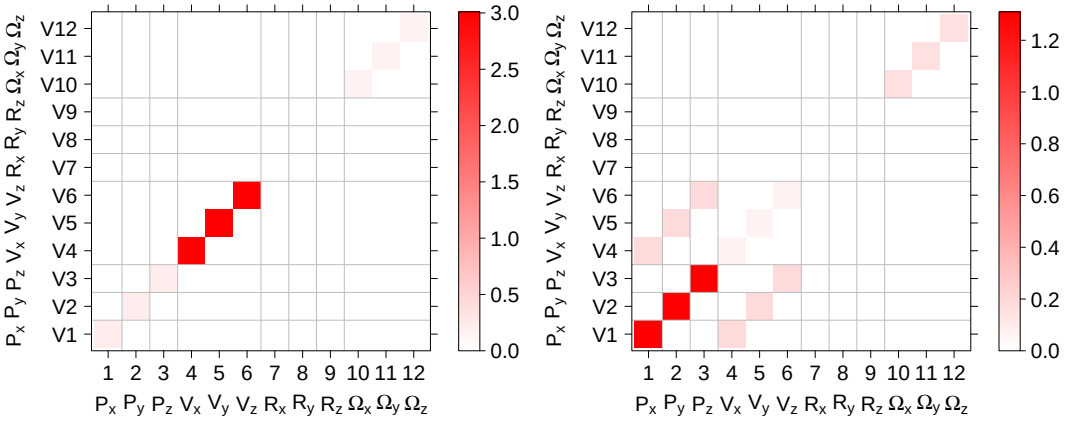


(b) Orientation using FAMM

Figure 6.26: Estimated rotations for CMM and FAMM for dataset 5. Colours indicate the type of the motion model used to estimate the orientation.

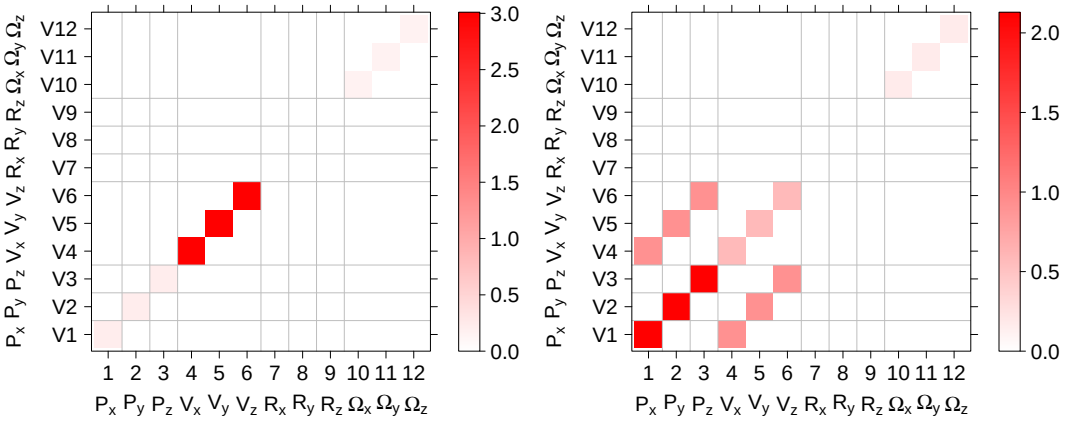
The aim of using multiple motion models was to decrease the uncertainty in the filter due to user motion, which is not always predictable. This observed decrease is mainly due to the selection of the most appropriate motion model, better fitting the measurements providing more supporting evidence for the filter so that it is more certain of its current state. The state covariance matrix of a KF (Σ in Algorithm 1) includes this estimate of uncertainty as well as correlations between state vector (x) elements. The diagonal elements indicate the variances and off-diagonal ones store correlations [313]. This matrix uses the information provided to the filter through the Kalman gain (K) indirectly from the measurements.

The changes in the state covariance matrix are shown in Figures 6.27 to 6.31 where colours of the squares are associated with the magnitude of the matrix elements. The scales for colours in the diagonal elements for the state covariance matrix in the figures for FAMM indicate a decrease in system uncertainty when it is used instead of CMM.



(a) GPS–IMU with CMM

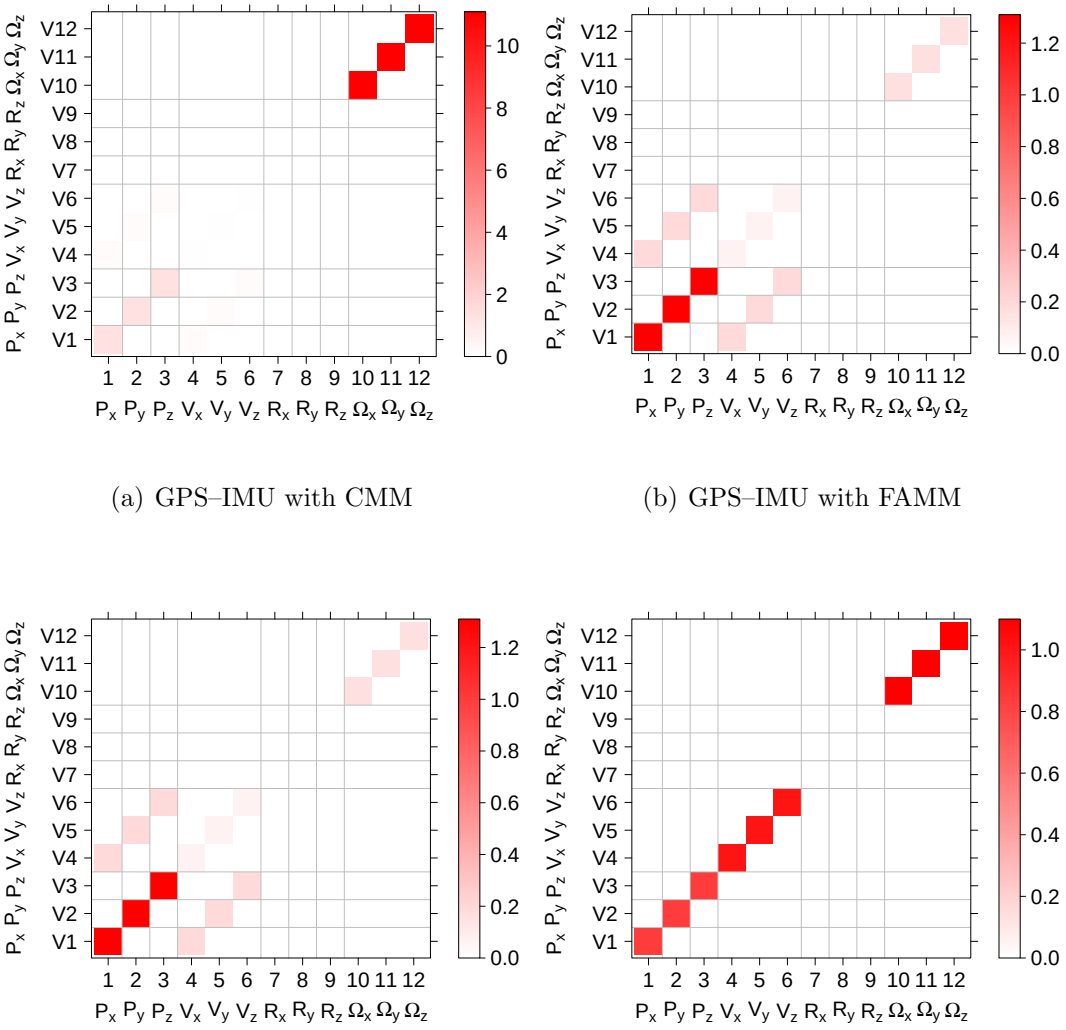
(b) GPS–IMU with FAMM



(c) Camera–GPS–IMU with CMM

(d) Camera–GPS–IMU with FAMM

Figure 6.27: Changes in the state covariances for dataset 1 when CMM and FAMM are employed for GPS–IMU and camera–GPS–IMU integration. Colours indicate the magnitude of the covariance matrix elements. Amount of uncertainty is illustrated by higher magnitudes (darker colours).



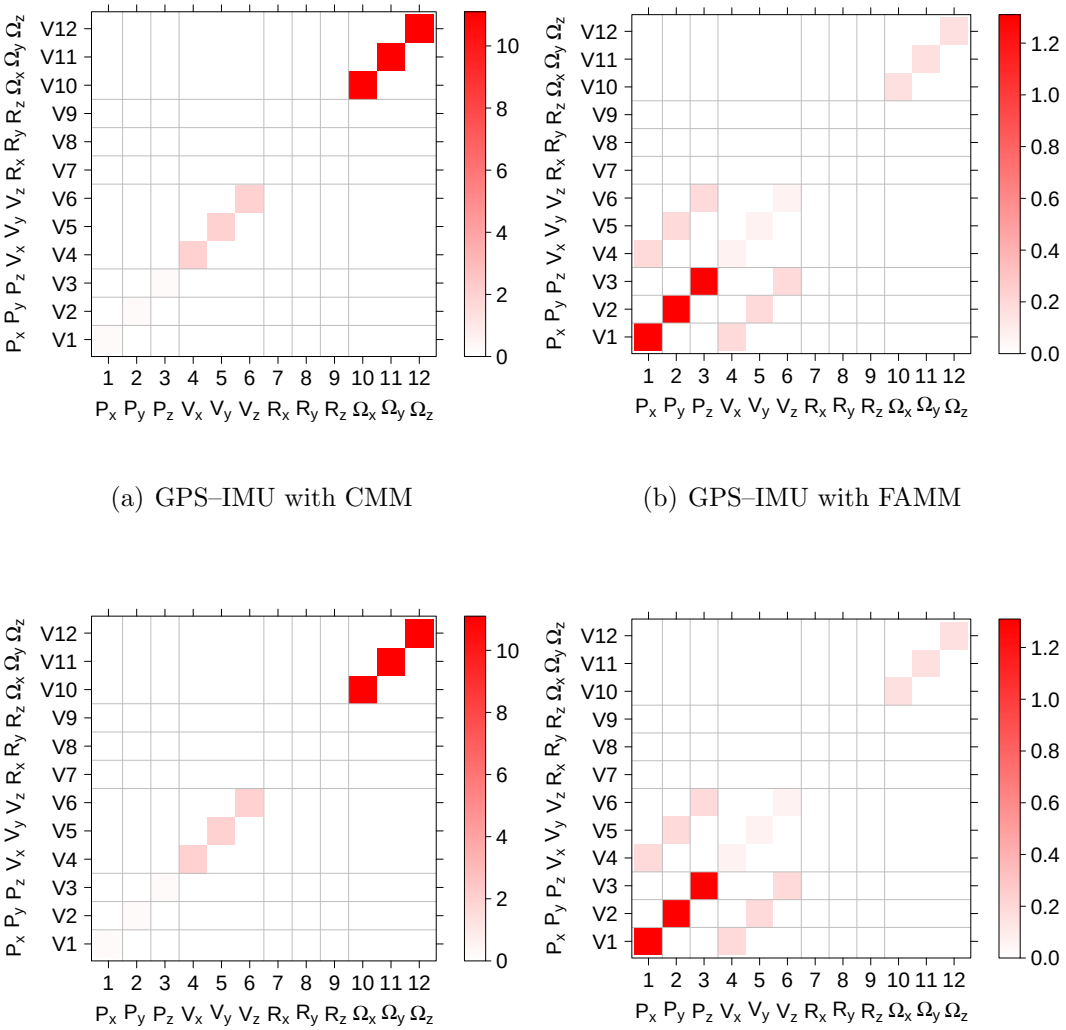
(a) GPS–IMU with CMM

(b) GPS–IMU with FAMM

(c) Camera–GPS–IMU with CMM

(d) Camera–GPS–IMU with FAMM

Figure 6.28: Changes in the state covariances for dataset 2 when CMM and FAMM are employed for GPS–IMU and camera–GPS–IMU integration. Colours indicate the magnitude of the covariance matrix elements. Amount of uncertainty is illustrated by higher magnitudes (darker colours).



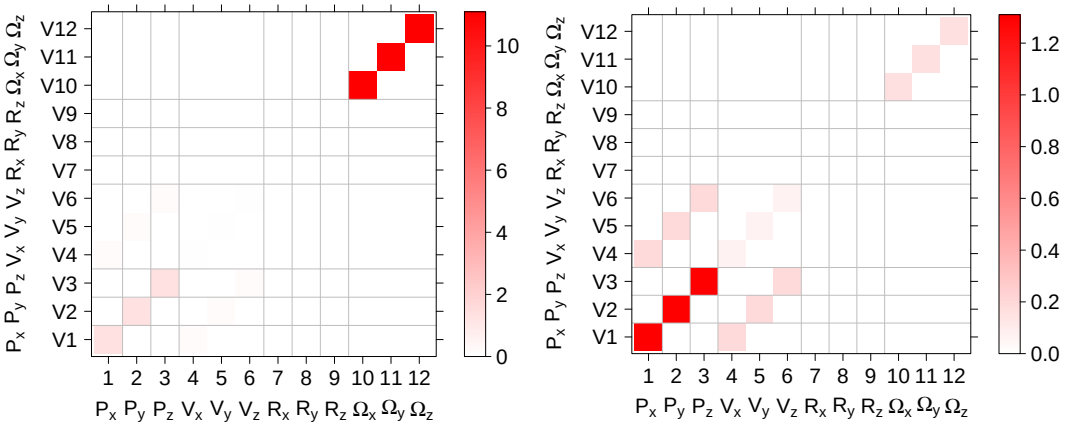
(a) GPS–IMU with CMM

(b) GPS–IMU with FAMM

(c) Camera–GPS–IMU with CMM

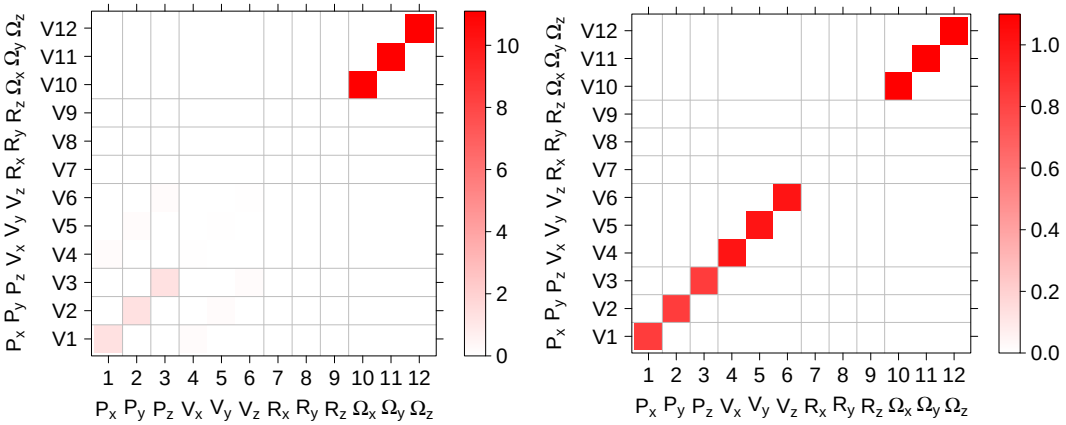
(d) Camera–GPS–IMU with FAMM

Figure 6.29: Changes in the state covariances for dataset 3 when CMM and FAMM are employed for GPS–IMU and camera–GPS–IMU integration. Colours indicate the magnitude of the covariance matrix elements. Amount of uncertainty is illustrated by higher magnitudes (darker colours).



(a) GPS–IMU with CMM

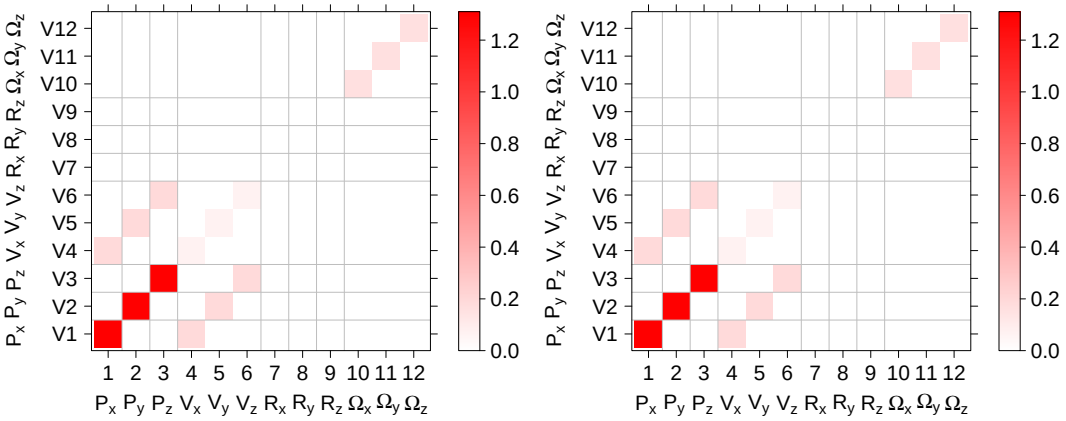
(b) GPS–IMU with FAMM



(c) Camera–GPS–IMU with CMM

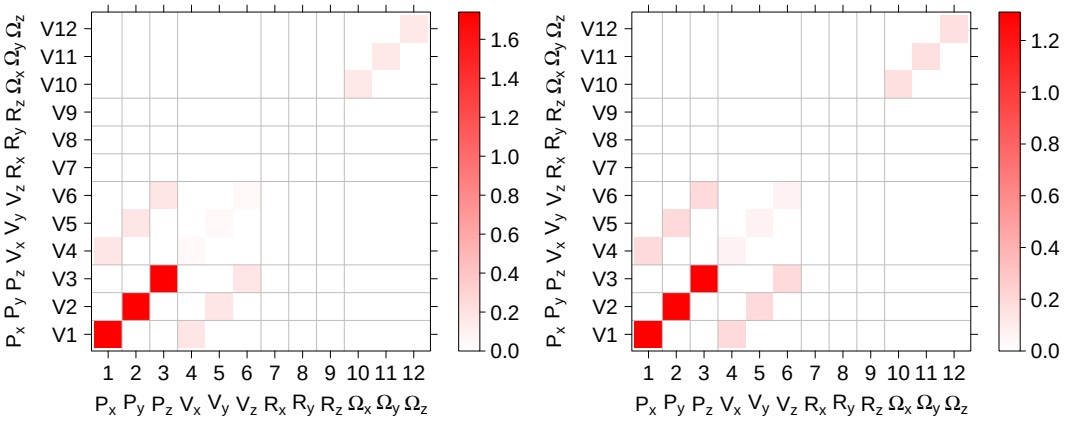
(d) Camera–GPS–IMU with FAMM

Figure 6.30: Changes in the state covariances for dataset 4 when CMM and FAMM are employed for GPS–IMU and camera–GPS–IMU integration. Colours indicate the magnitude of the covariance matrix elements. Amount of uncertainty is illustrated by higher magnitudes (darker colours).



(a) GPS–IMU with CMM

(b) GPS–IMU with FAMM

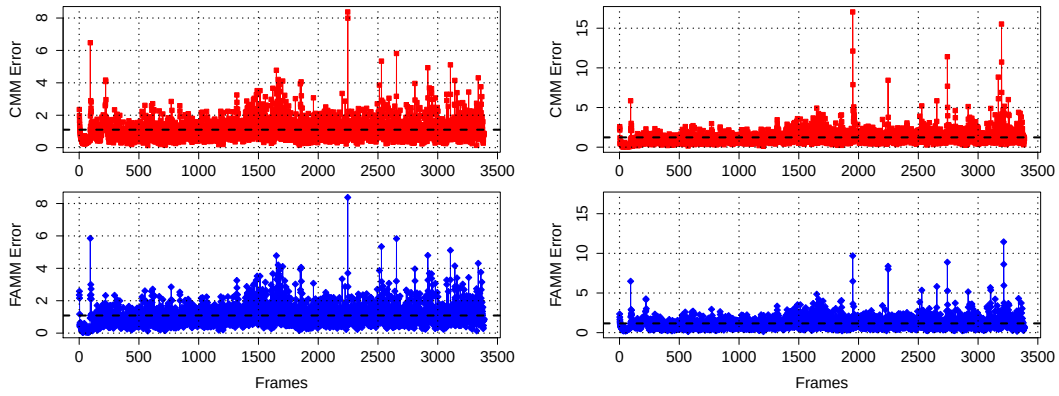


(c) Camera–GPS–IMU with CMM

(d) Camera–GPS–IMU with FAMM

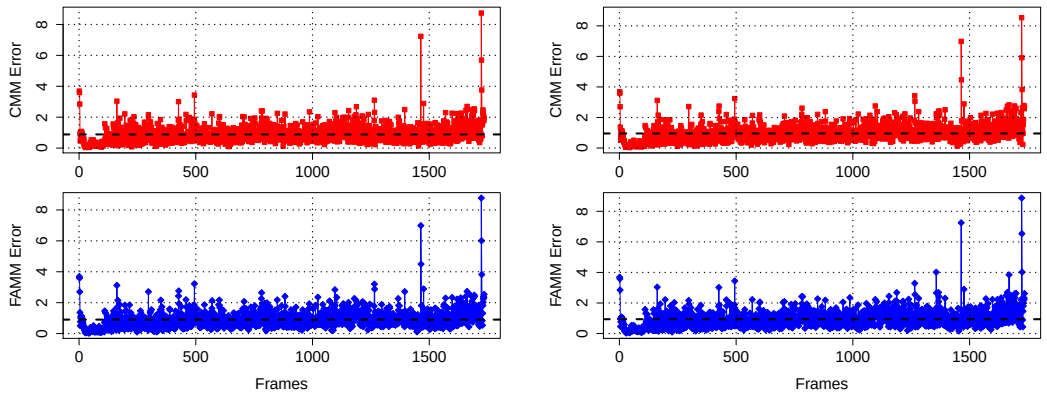
Figure 6.31: Changes in the state covariances for dataset 5 when CMM and FAMM are employed for GPS–IMU and camera–GPS–IMU integration. Colours indicate the magnitude of the covariance matrix elements. Amount of uncertainty is illustrated by higher magnitudes (darker colours).

It is also known that the state covariance matrix (Σ) is an approximation and not an actual error [314]. For this reason, the filter errors are also shown in Figures 6.32 to 6.36. Note that these errors are an indicator of the difference between the filter predictions and the actual values of the measurements, not the error calculation for ground truth data, which is shown in Figures 6.17 to 6.21.



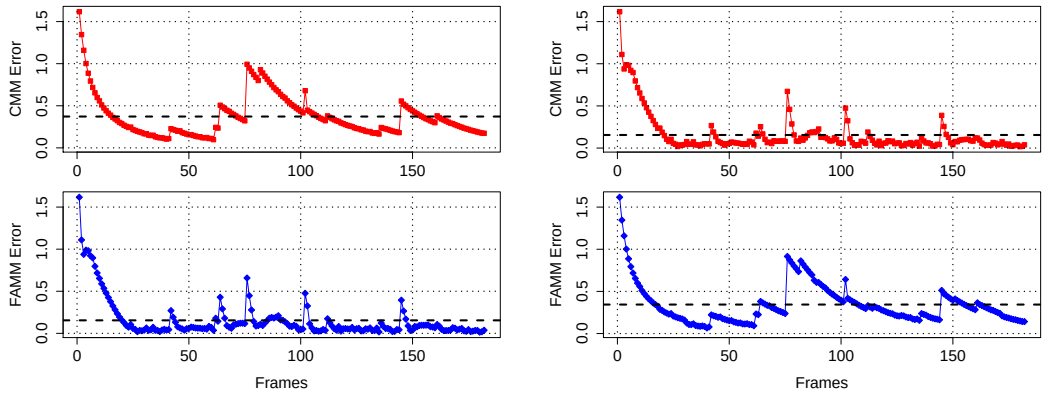
(a) Filter error using GPS and IMU (b) Filter error using GPS, camera and IMU

Figure 6.32: Filter errors for dataset 1 for CMM and FAMM. The mean error is shown with the dashed line.



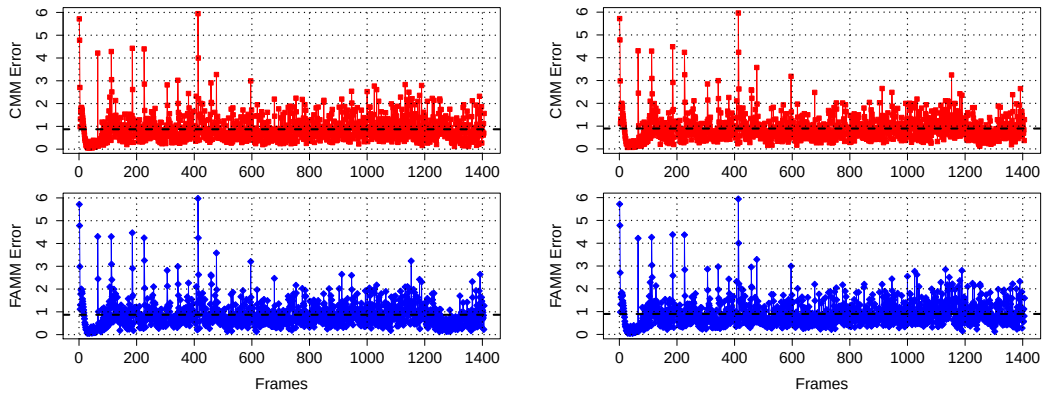
(a) Filter error using GPS and IMU (b) Filter error using GPS, camera and IMU

Figure 6.33: Filter errors for dataset 2 for CMM and FAMM. The mean error is shown with the dashed line.



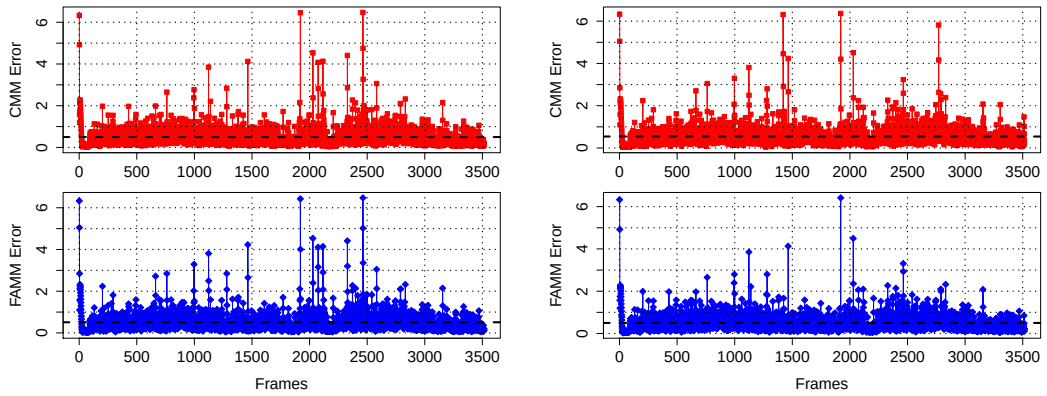
(a) Filter error using GPS and IMU (b) Filter error using GPS, camera and IMU

Figure 6.34: Filter errors for dataset 3 for CMM and FAMM. The mean error is shown with the dashed line.



(a) Filter error using GPS and IMU (b) Filter error using GPS, camera and IMU

Figure 6.35: Filter errors for dataset 4 for CMM and FAMM. The mean error is shown with the dashed line.



(a) Filter error using GPS and IMU (b) Filter error using GPS, camera and IMU

Figure 6.36: Filter errors for dataset 5 for CMM and FAMM. The mean error is shown with the dashed line.

Further to Figures 6.32 to 6.36, a decrease in the filter error is visible when FAMM is employed to choose the motion model; see also Tables 6.6 and 6.7.

Table 6.6: Mean and standard deviation of the filter error for the integration of GPS and IMU. Arrows indicate an increase/decrease when the FAMM is employed.

		CMM		**FAMM**		
Dataset	**Size**	**Mean Err.**	**Std. Err.**	**Mean Err.**	**Std. Err.**	
1	3388	1.106497	0.638612	1.087659	0.633043	↓
2	1735	0.884078	0.554315	0.903469	0.563023	↑
3	182	0.372074	0.247346	0.154518	0.232001	↓
4	1406	0.872035	0.573797	0.871161	0.569474	↓
5	3515	0.497994	0.430623	0.511027	0.447664	↑

Table 6.7: Mean and standard deviation of the filter error for the integration of camera, GPS and IMU. Arrows indicate an increase/decrease when the FAMM is employed.

		CMM		**FAMM**		
Dataset	**Size**	**Mean Err.**	**Std. Err.**	**Mean Err.**	**Std. Err.**	
1	3388	1.234321	0.885747	1.168871	0.732295	↓
2	1735	0.953222	0.576215	0.937526	0.581637	↓
3	182	0.155107	0.230386	0.343855	0.240241	↑
4	1406	0.894683	0.563316	0.897089	0.569789	↑
5	3515	0.543074	0.442076	0.498571	0.384069	↓

It can be seen that using an adaptive motion model (FAMM) decreases the error in general. It is also interesting to see that there are differences in errors when the fusion filter used estimates from different sensors and it can be seen that this error increased when the integration of camera, GPS and IMU is used for almost all datasets due to an additional source of noise. Note that the calculated error here is the filter error as a measure of filter convergence –not the ground truth positional error. The performance of the developed filter using three sensors in positional accuracy over the conventional GPS-IMU fusion can be seen clearly in the path

plots. One exception to this is the decrease for dataset 5 when FAMM was used. For this dataset, the accuracy of the filter was also obvious from the estimated path (Figure 6.21(e)). For the stationary dataset (3), the error decreased when the FAMM was use in a GPS–IMU integration but increased in the case of addition of the camera estimates to these two sensors. This is suspected to be due to the large baseline requirement (see section 4.4.1) for the two-view motion estimation algorithm, which is not possible when there is no motion.

6.6 Remarks

This chapter presented a tracking system using a fusion of the motion estimates from a camera, GPS receiver and IMU sensor within a KF framework and employing FAMM in order to reduce the filter error.

The discussion started by presenting the details about the sensors used in the experiments and described the sources of error that cause problems in tracking. Then, the methods used for obtaining motion (*i.e.* position and orientation) estimates from these sensors were described.

A sensor fusion algorithm employing these three sensors was presented in order to overcome the tracking errors (the static and dynamic errors mentioned earlier). The filter consisted of a simple state consisting of elements for position and orientation. Initially, this filter used a simple transition function. The motion estimate from the camera was applied to the GPS position estimate and this was interpolated by the IMU estimate in order to provide a continuous and smooth navigation. The estimate for orientation was obtained using the IMU filter [6]. The estimations from these sensors were calculated in different threads for performance.

The initial transition function was later updated by adaptive motion models which worked using fuzzy rules defining which motion model will be employed. These adaptive motion models had two parts, for the calculating the transition for the position and orientation estimates separately.

The results showed that the integration of the camera with GPS and IMU sensors provided more accurate results for tracking than a conventional GPS–IMU sensor fusion. From chapter 4, the vision-based algorithm was capturing the overall motion; however, fine detail was missing in the motion estimate. Furthermore, motion estimation was not accurate in cases of fast movements or cases when there is no motion. For the GPS, position estimate was erroneous and not accurate. IMU was accurate for a very short term, then drift was becoming a problem.

When the three sensors were used together, these problems were significantly reduced. The motion estimates from the camera reduced the accuracy problems for the GPS. This was further improved by using the IMU so that fine detail of the motion could also be captured. This integration of several sensors solved the problems related to static errors related to the accuracy of sensors.

Making use of the multiple threads allowed a better utilisation of the available resources. A second advantage of this design is that it helped reduce the dynamic errors, due to the end-to-end system delay, by providing a better frame rate.

This work also showed that multiple-motion model sensor fusion can be achieved by utilising KF innovation together with a fuzzy rule-base. The results show that the use of fuzzy adaptive motion models can reduce the filter error and prevent divergence. It is clear that selection of the appropriate motion model depending on user's speed improves the accuracy of KF for tracking applications such as the ones presented in chapter 7.

CHAPTER 7

CULTURAL HERITAGE APPLICATIONS

Developments in multimedia technology facilitate the learning experience in cultural heritage [315] with the aid of improved user interaction methods. Models or virtual tours of reconstructions of archeological sites (*e.g.* [165], [167]) provide an entertaining means of learning. However, *ex situ* reconstructions such as models and movies are difficult to visualize in the context of the archaeological remains.

One application of AR in cultural heritage [163, 166] can solve this problem by introducing *in situ* reconstructions that enrich the visiting experience of a heritage site with 3D models of ancient buildings. Another attractive property of AR reconstructions is that they can be rendered *in situ* with little or no physical disturbance to the ruins or artefacts (see section 2.6.1), even though such systems may take a significant time to develop [316].

There are several forms that AR reconstructions may take. Some applications allow the user to view reconstruction of ancient buildings by standing at a fixed viewpoint as in [163,164], while others allow navigating within a heritage site [317]. There are also studies which insert animated human models in order to emphasize the social value of a particular part of a building [318].

Kinect has also been used in the context of cultural heritage. For instance, Remondino [170] presented a review of using different types of imaging and depth sensors, including Kinect, to perform 3D scanning of archaeological objects for digital recording, historical documentation and preservation of cultural heritage. Richards-Rissetto *et al.* [171] used Kinect's body motion detection features to perform navigation in a 3D reconstructed model of an ancient Mayan city.

The principal problem faced in AR applications concerns the accuracy of user tracking and, consequently, the registration of 3D models with real-world features [11, 143, 212, 319]. When the 3D structure of the environment (feature positions, and internal/external parameters of the camera, *etc.*) is known, sufficient accuracy can be obtained to match virtual and real objects seamlessly; as a rule of thumb, this involves identifying the direction of gaze to $\sim 1^\circ$ in azimuth and elevation, and locating the position to $\sim 0.1\,\mathrm{m}$ [11].

A second important consideration is the frame rate. This relates to the dynamic errors [143] when the AR system cannot render the graphics quickly in accordance with the user's motion and most recent viewpoint (see section 2.8).

With these problems in mind, this chapter makes use of the algorithms presented in chapters 5 and 6 for three different exemplar AR applications aimed for use in a cultural heritage context. The discussion starts with explaining the stages of modelling of the 3D models used in the applications. These stages included generating the 3D model itself using different modelling techniques (*e.g.*

using profiles, Boolean operations, *etc.*), then optimizing the models to reduce the number of vertices and increase rendering speed, then finally wrapping the 3D models with textures for a more realistic look. The chapter continues the discussion by describing the rendering pipeline in section 7.2, including the game engine used for rendering and its scene graph structure, the approach used to render 3D models over the images acquired by the camera; and audio handling which is used in one of the three applications.

All of the applications of this chapter display the environment in an egocentric view [160] similar to First Person Shooter (FPS) games. The first application, presented in section 7.3, is an *in situ* augmentation which calculates the pose of the camera using 3D–2D correspondences using the depth information obtained from the Kinect sensor. Following this, rectangular features similar to columns found in the environment are augmented with synthetic column models. The application also shows that Kinect's skeleton tracking features can be used to augment the appearance of humans, so that they can be clothed in a way that matches their surroundings, in particular with *galea*[1], toga and sword. Existing AR systems in the cultural heritage domain do not attempt the latter, yet the author considers to be essential if participants are truly to experience a sense of presence following the ideas presented in [4].

The chapter presents a second application in section 7.4 which displays the State *Agora*[2] of ancient Ephesus by employing the tracking system developed in chapter 6. With this application, a user can walk inside the large building.

As a final application, section 7.5 describes an AR game in which the user aims to collect all reward items in the shortest time in order to obtain the highest

[1] A *galea* is a Roman military helmet.

[2] An *agora* is a public gathering place where discussions on politics, religion and commerce took place in ancient Greek cities.

score. This game can also be used together with the second application, giving a new aspect to cultural heritage applications of AR.

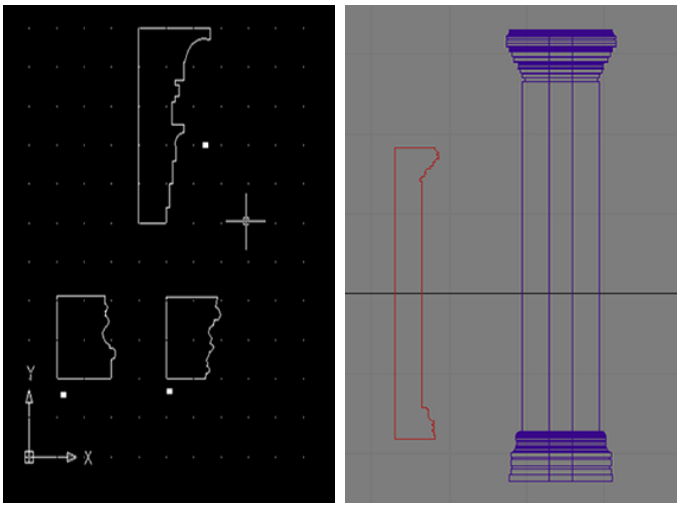
7.1 Generating 3D models

For any application making use of 3D models, creation of these models constitute a significant part of its development. This process is involved, especially if one is creating models for a cultural heritage application since fidelity to the original building structures [168] is an important consideration. This is followed by a stage called *optimization* in which the number of vertices and faces used to create the model are reduced as much as possible, since this will significantly affect the time spent rendering the models. The final stage includes 'wrapping' the 3D models with a texture for a more realistic appearance. Details of these stages are described below.

7.1.1 Modelling

Models of ancient columns and other building parts were created using 3D Max [320]. Buildings were modelled [167] in accordance with reconstruction images prepared by archaeologists [1], [2]. AutoCAD [321] was used to draw 2D profiles of buildings and other structures such as columns, as shown in Figure 7.1(a). These profiles were later exported to 3D Max where a number of different approaches can be used for modelling. Some of the 3D models were created using modifiers such as *bevel profile* which rotates the given profile around a boundary (*e.g.* square, circle *etc.*) to produce a 3D object (Figure 7.1(b)).

Buildings similar to houses can be created using the *extrude* modifier. This is done by first drawing closed lines that represent the walls of the building. These



(a) Column profiles (b) 3D column created from the profiles

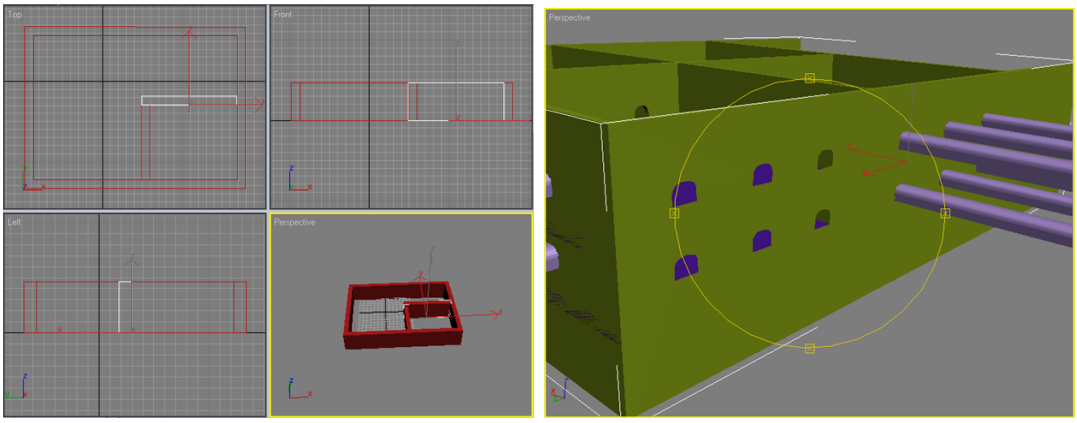
Figure 7.1: Creating 3D models of the columns in front of the Hadrian temple by spinning its profile around a circle based on the structure of the ruins and the reconstruction image in Figure 1.1.

lines are then extruded to yield solid walls as in Figure 7.2(a). Boolean operations (*e.g.* subtraction, union) are utilized to create the building windows and doors. Window and door profiles were again extruded to create 3D objects and then these are subtracted from the walls as depicted in Figure 7.2(b).

After creating models, the next step is reducing the number of faces used in the models for performance improvement.

7.1.2 Optimization

3D models having a large number of vertices (and hence faces) reduce the frame rate of a real-time application. For this reason, the models created in the previous section were optimized using the *MultiRes* modifier, which works by first computing the number of vertices and faces in a model, then allows the user to eliminate



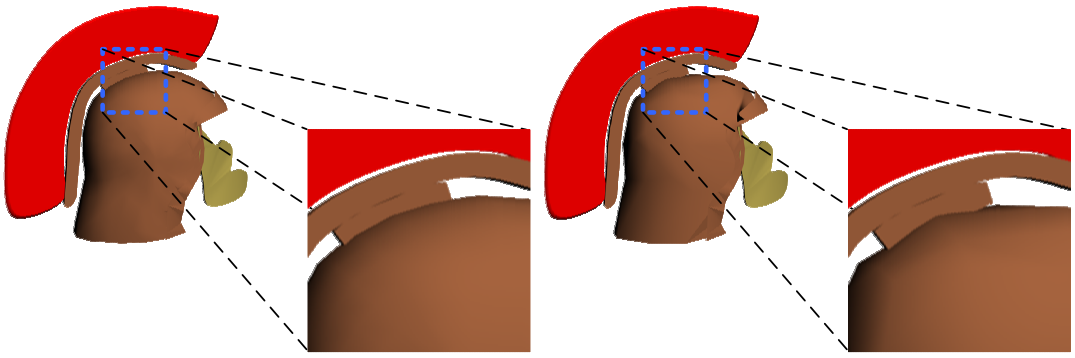
(a) Extrusion from profiles (b) Boolean subtraction

Figure 7.2: Methods for creating buildings. In (a), walls of a building can be drawn in form of lines and then these lines can be converted into 3D walls. Using these walls, blocks (shown with purple objects in (b)) can be subtracted in order to create the windows for the building.

some of them manually. This method proved to be effective for 3D models consisting of thousands of faces. For instance, for the spherical portion of the helmet model of Figure 7.3, the overall look can be retained with only 60 vertices (only 16.95% of the original 354 vertices) and 97 of the initial 657 faces, though some fine detail has been sacrificed in Figure 7.3(b).

The optimization process had a more significant improvement in the application displaying the state agora model, presented later in section 7.4 (Figures 7.15 and 7.16), where a large number of columns were present with a cylindrical shape and added detail, especially in the capital (top) section of the columns. Optimization results for different parts of the complete model are presented in Table 7.1.

The optimization stage was followed by preparing textures for use on the models.



(a) Original model (b) Optimized model

Figure 7.3: Effect of optimization on the helmet model, artefacts are noticeable on closer examination in (b)

Table 7.1: Optimization results for parts of the State Agora model. **Vertices** and **Faces** indicate the number of vertices and faces after the optimization, where as **Max. Vertices** and **Max. Faces** denote the numbers before optimization. **Decrease** shows the percentage of optimization in the number of vertices.

Part	**Vertices**	**Max. Vertices**	**Faces**	**Max. Faces**	**Decrease (%)**
Middle	4115	4328	8126	8552	4.921
Left	18804	25939	37616	51886	27.507
Right	5920	7040	11840	14080	15.909
Back	3040	3520	6080	7040	13.636
Front	13642	14949	25736	28303	8.743

7.1.3 Texture baking

A texture is a repeating detailed pattern that is applied to models. Texture mapping provide fine surface details (*e.g.* colour, reflection, normal vector perturbation ("bump mapping") *etc.*) [322] without requiring much effort to obtain them and hence produces a more realistic look in the final model.

A number of different textures (or materials) can be used for a model. When modelling a human for instance, the texture that will be used for the hair will be different than the texture used for the skin. Texture baking (also known as "rendering to texture") [323] is the process of creating a single texture map from multiple materials that have been applied to a model. This needed to be done since the game engine (see section 7.2.1) used for rendering required a single texture per model. There are several ways to perform this [324, 325] including using a light-map or incorporating shadows; the following method was found to produce the best results in 3D Max and is explained with a sample model.

To produce a single texture from several materials applied to a model, one first needs to make sure that the scene has adequate lighting, otherwise some faces of the model will remain in shadow. This will result in the texture elements not being prepared properly. A *skylight* was used in order to provide a global diffuse illumination in the scene [326] as shown in Figure 7.4.

As can be seen from the sample model, two different textures (roof and bricks), shown in Figure 7.5, were used. These are combined in a single texture map according to the outer faces of the model.

In order to prepare the final texture which is rendered for each of the individual faces of the model, the *Unwrap UVW* modifier is applied to the model to store the current material map, which defines how the texture must wrap around a model

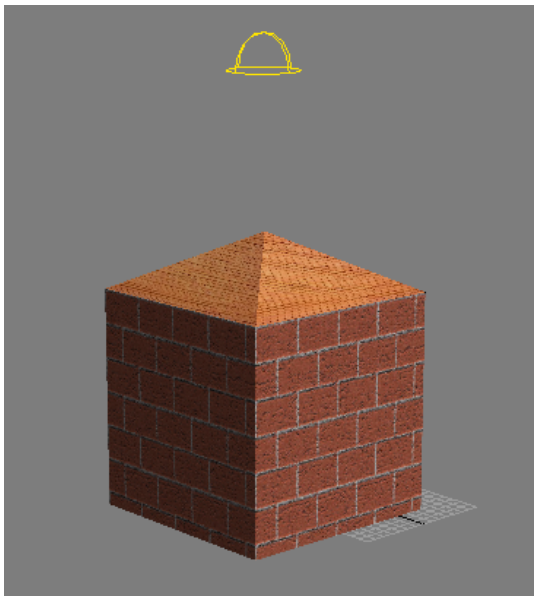
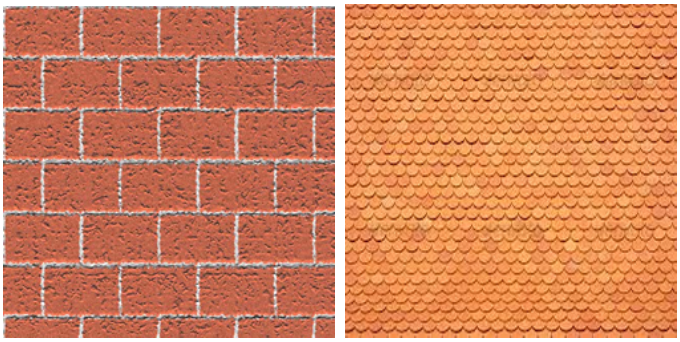


Figure 7.4: Sample model and sky light



(a) Brick (b) Roof tile

Figure 7.5: Textures used in the sample model of Figure 7.4

with a complex structure. After saving, all faces of the model are selected using the *face* element of the modifier as shown in Figure 7.6.

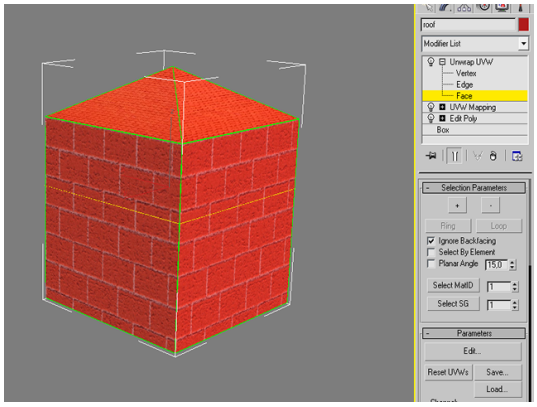


Figure 7.6: Selecting individual faces of the model

The next step is to unwrap (flatten) all the individual faces of the model as shown in Figure 7.7 and incorporate the material information into the output texture. A more complex model looks as in Figure 7.8, when the faces are unwrapped.

Now that the system knows how the faces are placed when the model is flattened, the next step is the actual rendering operation. This is done by the "Render to Texture" feature of the software. During the experiments (and considering the output by the game engine described in section 7.2), it was observed that the best results are obtained when only the diffuse element is used, rather than elements such as the complete map and lighting map. The output of the rendering is shown in Figure 7.9 in form of a single texture file in TGA format.

Later, this single texture was applied to the model again, by removing the previous texture material. Finally, the stored map must be applied to obtain the original look of the model before texture baking; then the final model is obtained as shown in Figure 7.10.

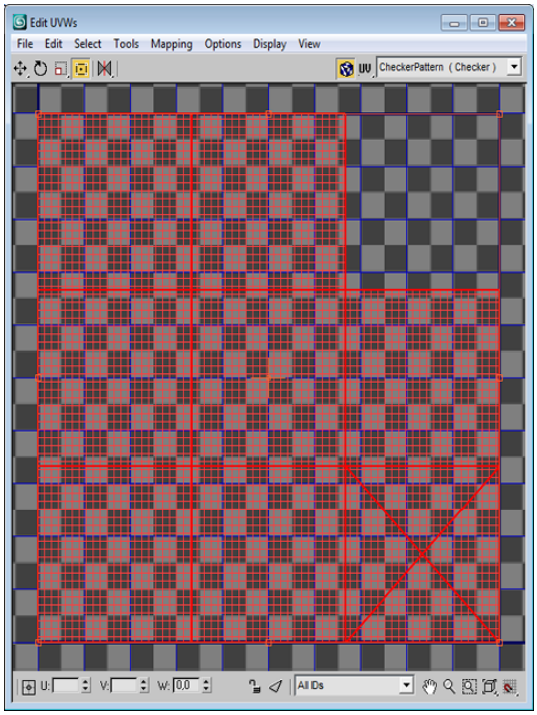
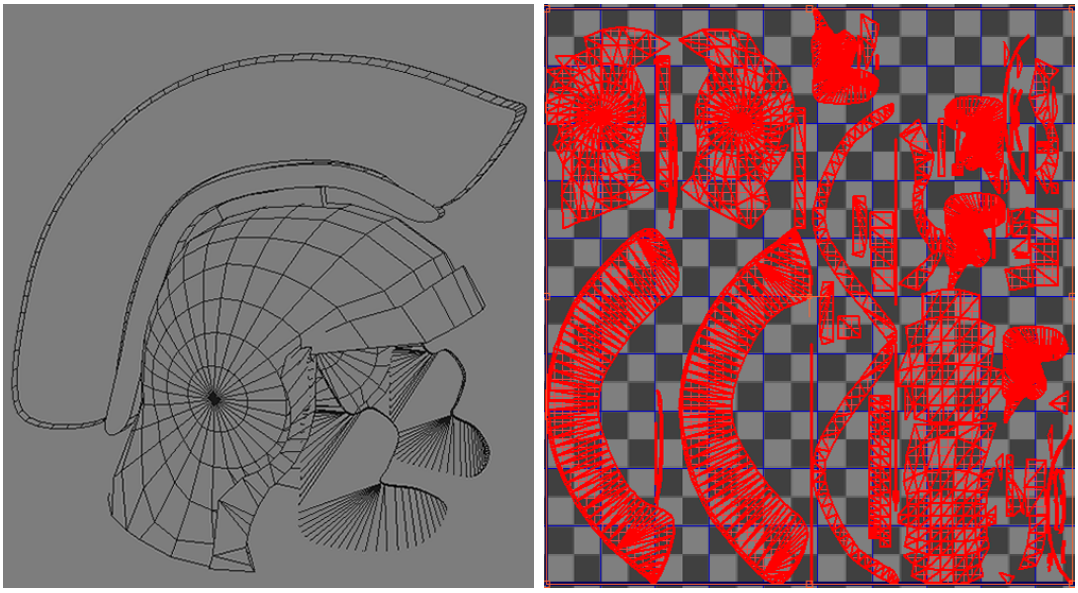


Figure 7.7: Unwrapping the faces of the sample model for texture baking



(a) Wireframe view of the model (b) Faces unwrapped

Figure 7.8: Unwrapping the faces of a complex model

Figure 7.9: Output texture

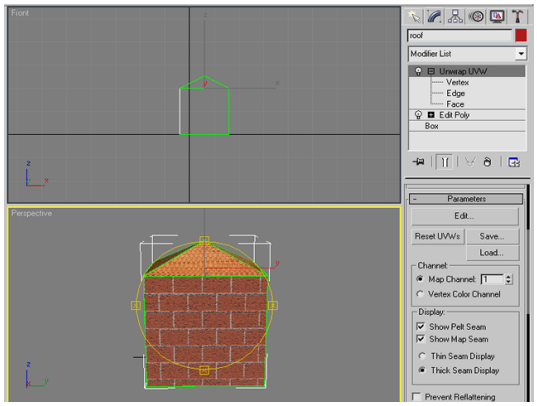


Figure 7.10: Final model with the baked texture

This operation was performed for all the models used in the applications presented in this chapter.

7.2 The Rendering Pipeline

Having 3D models ready, the next step is to prepare a rendering environment so that these models can be displayed on top of camera images acquired in real time according to the current position of the user. The following subsections will elaborate on how this is achieved.

7.2.1 Graphics engine and scene graph

Rendering is performed using the Irrlicht [327] game engine, which is an open source, real-time 3D engine. The engine uses a scene graph and provides support for a range of different file formats for loading and displaying the models created in the previous section.

In order to use the game engine, one first needs to initialize the graphics device parameters. An important point here is to set the dimensions of the graphics

devices the same size as the input images as this yields better results when graphics are rendered on the camera images.

The engine employs a scene manager, which is basically a scene graph; and this manager is responsible for rendering the content. The scene graph shown in Figure 7.11 has separate nodes for the camera, 3D meshes (models) and lighting.

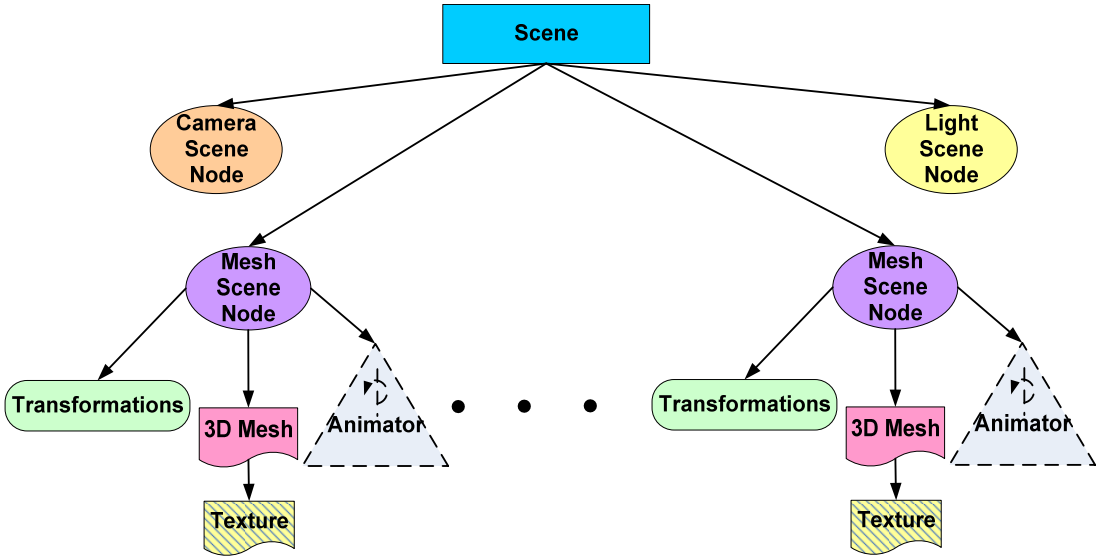


Figure 7.11: Scene graph structure. A scene can consist of many nodes including cameras, lights and meshes which can have child nodes for transformations, texture and animations.

The camera scene node is derived from Irrlicht's FPS camera. The main reason behind this is that this camera can update its target vector automatically when the position and orientation of the camera is updated using the camera pose-finding algorithms presented in chapters 5 and 6. It is also worth mentioning that this type of camera is normally handled by mouse input; this feature of the camera was disabled so that the update will only be based on the results of the algorithms presented.

The second important component of the scene graph is the mesh scene node,

which stores information about a 3D model. Each model has *mesh* and *texture* files created using the procedure described in section 7.1. The *transformation* element of the mesh scene node describes the absolute position and orientation of the model in the scene. The meshes used in the applications can be classified into two groups, namely static and animated meshes. For instance, building models are static meshes whereas coins in the AR game are animated meshes. Meshes in the latter group have an optional *animator* element. This element allows the meshes to rotate about their axes autonomously as an additional effect.

7.2.2 Rendering on camera images

When using optical-see-through displays, images from the real environment are obtained automatically using a mirror in the display device and when the graphics are rendered, the augmentation comes at no cost [148]. However, when using video-see-through displays or a camera as the input source, as in this work, the images of the real environment must also be rendered together with other graphics in order to produce the final image. This required a common format which can then be used to view both images, and the approach adopted was to convert camera images to the texture format of the game engine. Raw pixel data of the source image was copied into the format of the target image. At each call to the function drawing the whole scene, first the camera image is rendered and then the game elements are rendered on top of it.

7.2.3 Audio

The AR game presented in section 7.5 plays a simple sound when the user reaches and collects an item. These sounds were played using Simple DirectMedia Layer

(SDL), a multimedia library which is used to access low-level devices such as the graphics card or the audio driver. This is achieved by first initializing the library for its audio subsystem. Next, the SDL mixer is initialized using the frequency (44100), format (AUDIO_S16SYS), number of channels (2) and chunk size (2048) parameters. Finally, the audio file in WAVeform audio file (WAV) format is loaded.

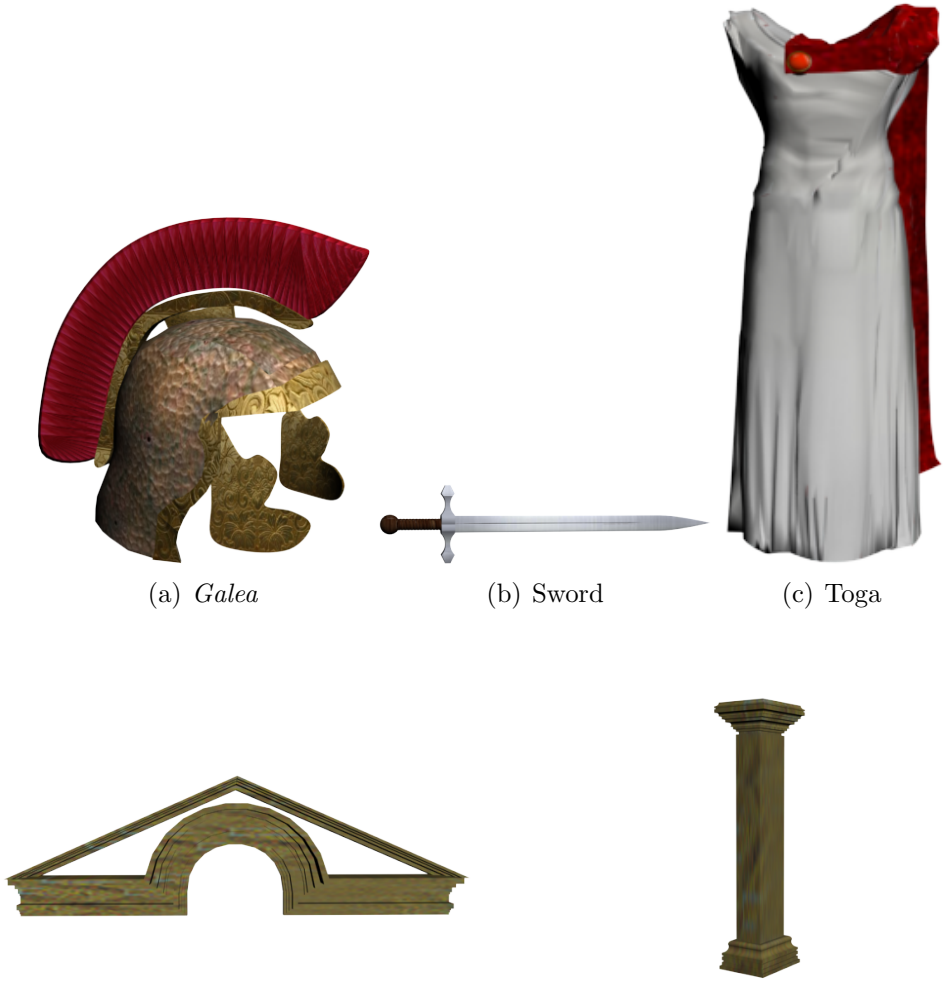
7.3 Application I: Kinect-Derived *In Situ* Augmentation

In the first AR application developed for cultural heritage, objects and users in the environment are augmented using column models and clothing appropriate to a specific age's fashion trends respectively.

The viewpoint required for augmentation is calculated using the approach described in chapter 5 which uses the Kinect sensor to obtain 3D–2D feature correspondences. These correspondences are used to obtain an initial estimate of the camera pose and then this initial estimate is refined using a KF for additional stability. With the viewpoint information obtained, rectangular features in the image are identified and augmented with columns: this is a common requirement in cultural heritage reconstructions in the eastern Mediterranean, as shown in Figure 1.1.

The models created for this application are presented in Figure 7.12. As described in section 7.1.2, the optimization step resulted in a $10-25\%$ reduction in the number of faces for the models. Usually, the more complex the model is, the higher the proportion of faces that can be removed without a noticeable change in appearance; here, these optimizations were most efficient in terms of remov-

ing faces for the *galea* model (Figure 7.12(a)) due to its roughly spherical shape (many faces are required for a smooth surface, hence many can be removed during optimization) whereas the column (Figure 7.12(e)) was unable to accommodate the removal of many vertices without losing detail of the capital (top) or pediment (base) which can be seen in the column profiles in Figure 7.1(a).



(a) *Galea* (b) Sword (c) Toga

(d) Top arch (e) Column

Figure 7.12: Models created for the *in situ* augmentation application

To illustrate the augmentation algorithms presented above, an application was developed to augment rectangular regions of a specific size and aspect ratio in the Kinect imagery. When the camera position and orientation had been found and the centres of suitable rectangles identified, the 3D column models described in section 7.1 were rendered in front of them. When rectangles were found at a particular distance apart, a further model could be placed above the columns to form an arch, as shown in Figure 7.13.

Figure 7.13: Augmenting columns over rectangles

The result of augmenting users is shown in Figure 7.14(a) for a single user and in Figure 7.14(b) for two users. The general effect is acceptable for toga and sword but minor registration errors are apparent for the galea model in the case of a single user. These registration errors become more severe when multiple users are present, and this appears to be due to the accuracy of skeleton tracking decreasing when the user is not centred in the field of view.

(a) Augmenting a single user with toga, galea and sword

(b) Augmenting two users

Figure 7.14: Augmenting participants

Due to the use of the FAST detector with the BRIEF descriptor, the complete augmentation process which can run at video rates (25fps).

7.4 Application II: A Visit to Ancient Ephesus

The second application presented here allows a user walk inside a large AR model of the State *Agora*, shown in Figure 7.15, in the ancient city of Ephesus, located in Turkey. The application makes use of the localization algorithm developed in chapter 6 to allow a user's motion in real world to be reflected in the application.

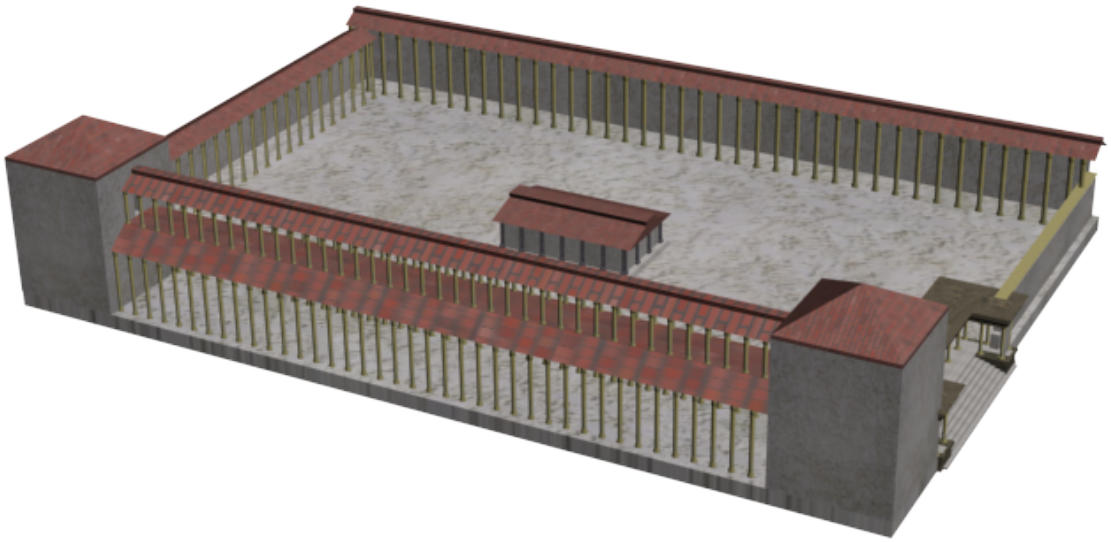


Figure 7.15: State *Agora* model

As can be seen, the model includes a large number of columns which follow the Ionic design [328] (*i.e.* with scrolls on the capitals). Such a high number of models with significant amount of detail add to the time required for rendering the scene. Initial versions of the applications ran at 2–3fps, which is unsatisfactory. The application's frame rate benefited from the following improvements:

1. Use of different threads for the module working for the vision-based algorithm (see the detailed discussion in section 6.3.2) and the module handling the GPS and IMU inputs to separate them from the fusion algorithms and the augmentation part have significantly improved the frame rate.

2. Optimization of different parts of the model, as shown in Table 7.1, have reduced the number of faces to be drawn facilitating a higher frame rate. Especially columns which are abundant in the scene and have fine detail due to the rolled structure of the capital were introducing a significant number of faces before the optimization was performed.

3. Division of the complete model into 5 parts as shown in Figure 7.16 helped, storing these parts separately in the scene graph. The rendering engine will not render the parts of the scene graph which are not inside camera's viewing frustum.

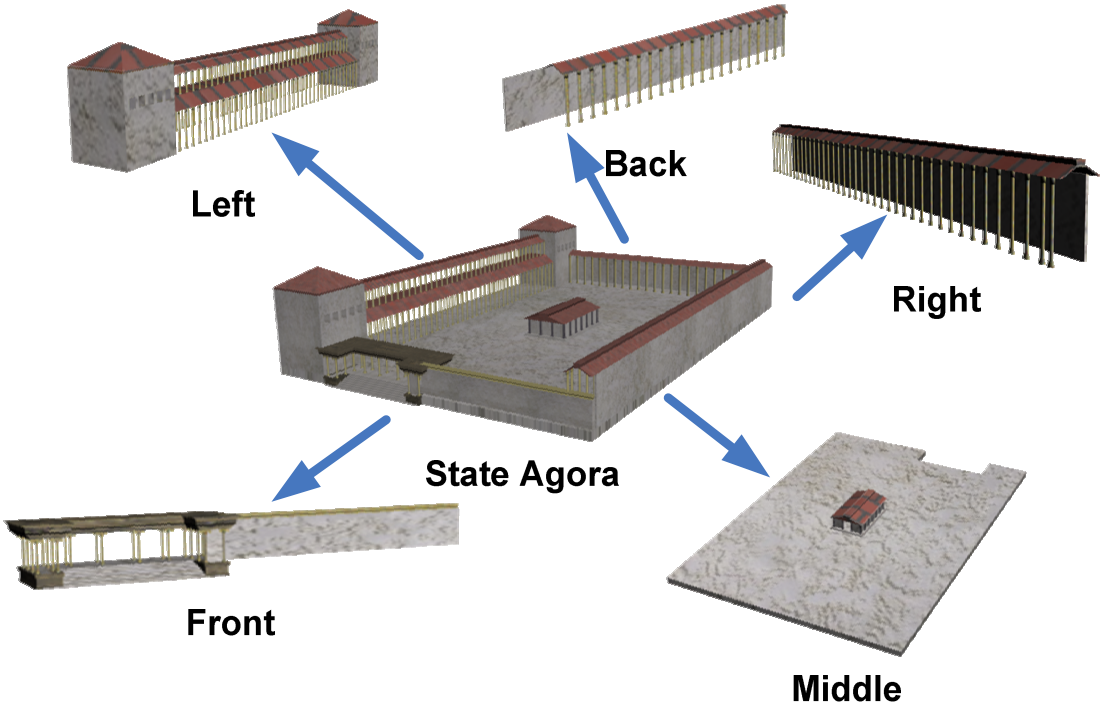


Figure 7.16: Division of the State *Agora* model for the scene graph

Following these improvements, the frame rate increased to 16fps which is still less than the de facto standards of 25–30fps yet providing an acceptable frame rate. The third optimization in particular allowed the frame rate to remain stable during display: frequent changes in frame rate according to the changing number of faces visible in the scene makes for a poor experience. The base of the middle part of Figure 7.16 was also removed in order to place the model on the ground.

Tracking is achieved on a laptop computer with the user wearing a helmet instrumented with a camera, GPS and an IMU as shown in Figure 7.17. The augmentations are displayed on the screen rather than an HMD.

Figure 7.17: User wearing the tracking system and displaying the models

Figure 7.18 presents views from the developed application. The user interface for this application is quite simple, just displaying the name of the model and the current frame rate.

(a) Entering through the gate

(b) Inside the *Agora*

Figure 7.18: Views from the AR application

7.5 Application III: An AR Game – Treasure Hunt

The final application is an AR game, which again works using the tracking system (chapter 6). The aim of the game is to collect items and direct the user to close a loop within the SLAM context, albeit unconsciously.

As with previous applications, the game presents an egocentric view of the environment. The rules of the game are quite simple: the user needs to reach and collect all the reward items available as quickly as possible. When he or she reaches an item, the score is incremented by an amount that depends on the type of item encountered. The game provides three types of items: small coins, large coins and a chest (Figure 7.19), with rewards of 10, 30 and 50 points respectively.

(a) Chest

(b) Coin

Figure 7.19: Models used in the AR game

After the game is initialized with the positions of all items set, the game loop starts. The coin models use the animator and they rotate about their axes while the chest models remain static.

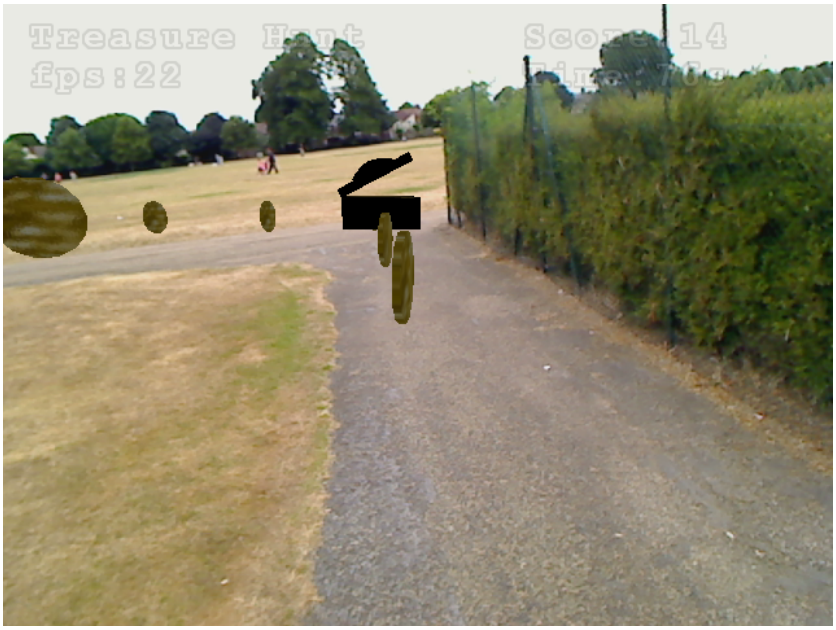
At each frame, the position of the user is checked against the item positions by

calculating the distance between them. If this distance is less than some threshold value (done so that there is some tolerance against positioning inaccuracies), then the score is updated, the item is set as "hit" and a sound file is played as described in section 7.2.3. The items collected by the user simply disappear. A timer is used for two purposes. First, it is constantly updated in the display to provide feedback to the user. It is also used to decay the score

$$score = initialScore \times c/time \tag{7.1}$$

where *score* is the final score to be added and *initialScore* correspond to the rewards mentioned above. The constant c is selected as 5.0 arbitrarily. This forces the user to collect the game tokens quickly.

Views from the AR game are presented in Figure 7.20. The game has an interface which displays the score and time passed making the game more challenging and hence interesting. Figure 7.20(b) shows that the game can also be played with the AR reconstruction of the State *Agora*. Note the frame rate of the game is higher when the game is displayed on its own (*i.e.* without the reconstruction).



(a) User collecting reward items

(b) The game can be played inside the State *Agora*.

Figure 7.20: Views from the AR game

7.6 Remarks

This chapter presented three simple AR applications for cultural heritage, using the user-tracking algorithms described in earlier chapters. The discussion started with the approaches used for creating the 3D models used in the applications: the modelling stages included generation of the model using different modelling techniques such as extrusion from 2D profiles or boolean operations. The next stage reduced the complexity of the models to improve rendering performance. The final stage was producing textures to give the models a more realistic look.

The rendering pipeline was presented in section 7.2, which consisted of three main elements. The first of these was the graphics engine and the scene graph which was responsible for the rendering of the 3D models. Secondly, the approach used for rendering the camera images as the background for augmentation was described: this converted the camera image to the texture format used by the engine. After the texture is drawn, the 3D models were overlaid. The last element of the rendering pipeline was the use of audio which was used to play simple sounds when users reached reward items in the AR game of section 7.5.

The chapter then presented three applications developed for cultural heritage. The first one (section 7.3) was an *in situ* augmentation application which overlaid colum-like objects with synthetic 3D columns and people with ancient clothing. The application used the *in situ* augmentation algorithm with Kinect described in section 5.3 which finds the camera pose with 3D–2D correspondences thanks to the depth sensor of Kinect. Rectangular structures found in the environment were augmented with columns. Using the human skeleton tracking features of Kinect users were also augmented with a toga, *galea* and sword.

The second application presented an AR reconstruction of the State *Agora* of

ancient Ephesus. A user, wearing the tracking system presented in section 6.3.3, can walk inside the model, allowing them experience the surroundings of ancient times. This application can also be used in conjunction with the third application presented in this chapter, in which the user tries to collect reward items such as coins or chests as quickly as possible.

From the three applications presented here, it is clear that AR has a great potential for cultural heritage. Using the tracking methods presented in chapters 5 and 6, the user position can be found with enough accuracy so that the augmentation results are satisfactory. It is also important to note that these methods do not require any set-up phase since they use the natural features extracted from the environment. On the other hand, the rendering method used here does not handle occlusions between the virtually inserted models and real world objects due to simple overlay technique used.

The lack of any set-up phase and this speed of processing are important practical considerations for installations in museums that are to be used without supervision, or for use in educational games. Indeed, it is certain that AR applications similar to ones presented here will improve the on-site learning experience [315] and provide people with an incentive to learn about their and other people's past and protect the historical artefacts and monuments as a memory of the past.

The next chapter will summarize the contributions of the thesis, conclude the thesis and present future directions.

CHAPTER 8

CONCLUDING REMARKS

This thesis aimed to develop user tracking methods for cultural heritage applications for indoor and outdoor environments. Mainly, a vision-based system (chapters 4 and 5) is used for these purposes in indoor environments but a sensor fusion approach (chapter 6) was found to provide better tracking results in outdoor environments. The conclusions from this work is presented in section 8.1 and further research is suggested in section 8.2.

8.1 Conclusion

Application of AR to cultural heritage is a fascinating research topic. It allows preserving the original building structures, already subject to wear and tear over hundreds of years, and provide an entertaining way of learning their history by seeing the original building structures instead of ruins.

Apart from creating accurate models of the ancient buildings, the most important part in developing such applications is performing the user tracking (*i.e.* finding their position and orientation in the environment) accurately.

An initial attempt to solve this problem was using a popular visual SLAM algorithm from the robotics community, as described in chapter 4. Indoor experiments (small environment, such as a desktop) have shown satisfactory accuracy but the method was not able to perform tracking in an outdoor environment. Several approaches were tried to overcome this problem; however, results were not at all satisfactory.

The keyframe based localization algorithm presented in chapter 4 used two-view geometry to estimate the motion of the user and achieved more promising results in terms of tracking. This method was able to track the overall motion but missing some fine details due to vision processing.

Considering the operation in outdoor environments, the aid of the GPS and IMU sensors was exploited in a sensor fusion framework in order to achieve more accurate tracking results in chapter 6. The sensor fusion algorithm combined the position and rotation measurements from the camera, GPS and IMU to produce a single estimate for the user pose. The additional data processing did not add to the computation time; indeed, it was reduced due to the design making use of multiple threads.

A further improvement was achieved in chapter 6 where a fuzzy rule-base was employed to find the motion model for the Kalman filter that best fits the actual measurements coming from the sensors. This allowed the system to capture the motion of the user more accurately for cases where he or she walks at varying speeds or stands, an important feature for a user walking around an ancient site and frequently stopping to examine the ruins.

The recently launched Kinect sensor was also exploited for both tracking the user, using its skeleton-tracking capabilities, and finding the camera position using natural features, using its depth sensor, in the environment for *in situ* augmentation as described in chapter 5. Such a setup deployed in exhibits will obviously improve the visiting experience in museums.

Using the user tracking system for outdoor environments of chapter 6 as well as the camera motion estimation and user tracking features of the Kinect sensor in chapter 5, three applications were presented in chapter 7. The first one was for viewing ancient columns augmented over natural features and augmenting users with clothing from ancient ages. The second application allowed the user to view a model of a State *Agora* and walk around it. The final application was a simple AR game in which the user has to collect all the reward items (coins and chests) in the same context as the second application.

Since vision processing is used in all of the user tracking algorithms presented in the thesis and homography estimation plays a crucial role in almost all similar vision related applications, a new measure for calculating coverage of the image features (which is known to affect the accuracy of the estimated homography) is presented in chapter 3. This measure used a robust metric that analyses the distribution of features across the image. A dozen feature detectors were evaluated and the effect of coverage was also demonstrated in a simple image-stitching application.

Azuma stated in 1997 that AR tracking for outdoors in real-time with required accuracy is an open problem [143]. Though more than a decade has passed after his statement, this problem still keeps its validity since AR requires high accuracy, low latency and low jitter [144]. Similarly in [192], it was stated that there was a great need for a self-tracker that can be used in natural outdoor environments as

well; however, a robust implementation of such a tracker was years away due to the challenges in finding robust features in natural environments [3].

The author believes that using the current processing power of mobile systems (*e.g.* a next-generation Raspberry Pi[1]), techniques (*e.g.* multiple cores) and sophisticated algorithms (*e.g.* fuzzy adaptive filtering) making use of several sensors, we are much closer to seeing applications following the "Total Augmentation" paradigm presented in chapter 1.

8.2 Future Work

The contributions presented in this thesis are as open to further improvement as any other piece of research. A number of directions will provide a general outline for future research. These can be listed as:

- A 3D feature can provide 4 measurements when its projection is used for two images (feature point correspondences between keyframes in chapter 4). The use of the tri-focal tensor [329] (using correspondences between three keyframes) can be investigated since this can provide 6 measurements per feature which can compensate for errors [31], though this approach is more computationally expensive.

- The fuzzy rule-base defined in chapter 6 currently has 81 rules in order to cover all possible transitions between different motion models. Although these rules are accessed efficiently since a direct mapping is used in the implementation, a further improvement can be achieved using rule reduction [127] to overcome the exponential growth of the rule-base, which is

[1] `http://www.raspberrypi.org/`

regarded as a weakness of fuzzy logic based systems.

- The AR application presented in chapter 7 can display a large ancient building along with a simple game making the application more interesting. This can be further improved by creating an animated/live environment using the AR agents presented [4] in order to provide information in cultural heritage context.

- The user tracking system of chapter 6 is designed for a single user but can be extended to a group tracking system in a leader-follower scenario, such as a tourist group guided by a tour guide. This may be used to distribute the processing over multiple users and provide better accuracy following the system designed for robots in [330].

- Genetic Programming (GP) [331] can be used to refine the output of a feature detector so that the resulting set of feature points can yield a more uniform coverage across the image according to the measure defined in chapter 3.

–The End–

BIBLIOGRAPHY

[1] W. Alzinger and A. Osterreichisches Archaologisches Institut (Vienna, *Die Ruinen von Ephesos*. Koska, 1972.

[2] F. Hueber, S. Erdemgil, and M. Buyukkolanci, *Ephesos*. Zaberns Bildbande zur Archaologie, von Zabern, 1997.

[3] E. Bostanci, N. Kanwal, S. Ehsan, and A. F. Clark, "User tracking methods for augmented reality," *International Journal of Computer Theory and Engineering*, vol. 5, no. 1, pp. 93–98, 2013.

[4] E. Bostanci and A. F. Clark, "Living the past in the future," in *International Conference on Intelligent Environments, International Workshop on Creative Science*, pp. 167–172, 2011.

[5] I. Barakonyi and D. Schmalstieg, "Augmented reality agents for user interface adaptation," *Computer Animation and Virtual Worlds*, vol. 19, pp. 23–35, 2008.

[6] S. Madgwick, R. Vaidyanathan, and A. Harrison, "An efficient orientation filter for inertial measurement units (IMUs) and magnetic angular rate and gravity (MARG) sensor arrays," tech. rep., Department of Mechanical Engineering, 2010.

[7] E. Bostanci, N. Kanwal, and A. F. Clark, "Feature coverage for better homography estimation: An application to image stitching," in *Proceedings of the IEEE International Conference on Systems, Signals And Image Processing*, 2012.

[8] E. Bostanci, N. Kanwal, and A. Clark, "Spatial statistics of image features for performance comparison," *Image Processing, IEEE Transactions on*, vol. 23, no. 1, pp. 153–162, 2013.

[9] E. Bostanci, A. Clark, and N. Kanwal, "Vision-based user tracking for outdoor augmented reality," in *Computers and Communications (ISCC), 2012 IEEE Symposium on*, pp. 566–568, 2012.

[10] E. Bostanci, N. Kanwal, and A. F. Clark, "Extracting planar features from Kinect sensor," in *Computer Science and Electronic Engineering Conference*, pp. 118–123, 2012.

[11] E. Bostanci, N. Kanwal, and A. F. Clark, "Kinect-derived augmentation of the real world for cultural heritage.," in *UKSim*, pp. 117–122, 2013.

[12] D. J. Felleman and D. C. V. Essen, "Distributed hierarchical processing in the primate cerebral cortex," *Cereb Cortex*, pp. 1–47, 1991.

[13] S. Chen and J. Weng, "State-based shoslif for indoor visual navigation," *Neural Networks, IEEE Transactions on*, vol. 11, no. 6, pp. 1300–1314, 2000.

[14] N. Karlsson, E. Di Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian, and M. Munich, "The vslam algorithm for robust localization and mapping," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 24–29, 2005.

[15] C. Zhao and G. Jiang, "Baseline detection and matching to vision-based navigation of agricultural robot," in *Wavelet Analysis and Pattern Recognition (ICWAPR), 2010 International Conference on*, pp. 44–48, 2010.

[16] B. Sinopoli, M. Micheli, G. Donato, and T. Koo, "Vision based navigation for an unmanned aerial vehicle," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 2, pp. 1757–1764, 2001.

[17] Y. F. Cao, M. Ding, L. K. Zhuang, Y. X. Cao, S. Y. Shen, and B. Wang, "Vision-based guidance, navigation and control for unmanned aerial vehicle landing," in *Applied Sciences and Technology (IBCAST), 2012 9th International Bhurban Conference on*, pp. 87–91, 2012.

[18] N. Thamrin, N. Arshad, R. Adnan, R. Sam, N. Razak, M. Misnan, and S. Mahmud, "Simultaneous localization and mapping based real-time inter-row tree tracking technique for unmanned aerial vehicle," in *Control System,*

Computing and Engineering (ICCSCE), 2012 IEEE International Conference on, pp. 322–327, 2012.

[19] P. Belhumeur, J. Hespanha, and D. Kriegman, "Eigenfaces vs. fisherfaces: recognition using class specific linear projection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 19, no. 7, pp. 711–720, 1997.

[20] M. Yang, D. Kriegman, and N. Ahuja, "Detecting faces in images: a survey," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 1, pp. 34–58, 2002.

[21] K. Nickels and S. Hutchinson, "Model-based tracking of complex articulated objects," *Robotics and Automation, IEEE Transactions on*, vol. 17, no. 1, pp. 28–36, 2001.

[22] E. Dominguez, C. Spinola, R. Luque, E. Palomo, and J. Muoz, "Object recognition and inspection in difficult industrial environments," in *Industrial Technology, 2006. ICIT 2006. IEEE International Conference on*, pp. 989–993, 2006.

[23] Y. Yoon, A. Kosaka, and A. Kak, "A new kalman-filter-based framework for fast and accurate visual tracking of rigid objects," *Robotics, IEEE Transactions on*, vol. 24, no. 5, pp. 1238–1251, 2008.

[24] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 25, no. 5, pp. 564–577, 2003.

[25] Y. Ma, S. Soatto, J. Kosecka, and S. S. S., *An Invitation to 3-D Vision: From Images to Geometric Models*. Springer, 2004.

[26] Y. Furukawa, B. Curless, S. Seitz, and R. Szeliski, "Reconstructing building interiors from images," in *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 80–87, 2009.

[27] Z. Zhang and H. S. Seah, "Cuda acceleration of 3d dynamic scene reconstruction and 3d motion estimation for motion capture," in *Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on*, pp. 284–291, 2012.

[28] A. Broggi, P. Grisleri, T. Graf, and M. Meinecke, "A software video stabilization system for automotive oriented applications," in *Vehicular Technology Conference, 2005. VTC 2005-Spring. 2005 IEEE 61st*, vol. 5, pp. 2760–2764, 2005.

[29] G. Puglisi and S. Battiato, "Fast block based local motion estimation for video stabilization," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, pp. 50–57, 2011.

[30] A. Fitzgibbon, "From the lab to the silver screen: computer vision and the art of special effects," in *3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004. Proceedings. 2nd International Symposium on*, pp. 716–716, 2004.

[31] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[32] D. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*. Pearson Education, Limited, 2011.

[33] J. Weng, P. Cohen, and M. Herniou, "Camera calibration with distortion models and accuracy evaluation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 14, no. 10, pp. 965–980, 1992.

[34] Z. Zhang, "A flexible new technique for camera calibration," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 11, pp. 1330–1334, 2000.

[35] J. Y. Bouguet, "Camera calibration toolbox for matlab." `http://www.vision.caltech.edu/bouguetj/calib_doc/index.html`, 2008. Last access: November, 2013.

[36] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pp. 1297–1304, 2011.

[37] D. Castro, J. Kannala, and J. Heikkila, "Accurate and practical calibration of a depth and color camera pair," in *International Conference on Computer Analysis of Images and Patterns*, 2011.

[38] N. Burrus, "Kinect calibration." `http://nicolas.burrus.name/index.php/Research/KinectCalibration`, 2012. Last access: November, 2013.

[39] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly, 2008.

[40] G. Borenstein, *Making Things See: 3D vision with Kinect, Processing, Arduino, and MakerBot*. O'Reilly Media, 2012.

[41] M. Zuliani, C. Kenney, and B. S. Manjunath, "A mathematical comparison of point detectors," in *IEEE Computer Vision and Pattern Recognition*, 2004.

[42] M. Shneier, "Extracting linear features from images using pyramids," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 12, pp. 569–572, 1982.

[43] G. R. Cross and A. K. Jain, "Markov random field texture models," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI, no. 1, pp. 25–39, 1983.

[44] J. Shi and J. Malik, "Normalized cuts and image segmentation," in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pp. 731–737, 1997.

[45] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 886–893, 2005.

[46] T. Tuytelaars and K. Mikolajczyk, "Local invariant feature detectors:A survey," *Foundations and Trends in Computer Graphics and Vision*, vol. 3, no. 3, pp. 177–280, 2008.

[47] C. Schmid, R. Mohr, and C. Bauckhage, "Evaluation of interest point detectors," *International Journal of Computer Vision*, pp. 151–172, 2000.

[48] O. M. Mozos, A. Gil, M. Ballesta, and O. Reinoso, "Interest point detectors for visual slam," *Current Topics in Artificial Intelligence*, vol. 4788, pp. 170–179, 2007.

[49] S. Smith and J. Brady, "Susan: A new approach to low level image processing," *International Journal of Computer Vision*, vol. 23, no. 1, pp. 45–78, 1997.

[50] C. Harris and M. Stephens, "A combined corner and edge detector," in *Alvey Vision Conference*, pp. 147–152, 1988.

[51] C. Tomasi and T. Kanade, "Detection and tracking of point features," *Carnegie Mellon University Technical Report*, 1991.

[52] J. Shi and C. Tomasi, "Good features to track," *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 593–600, 1994.

[53] T. Tuytelaars, L. Van Gool, L. Dhaene, and R. Koch, "Matching of affinely invariant regions for visual servoing," in *Proceedings of the International Conference on Robotics and Automation*, 1999.

[54] T. Tuytelaars and L. Van Gool, "Wide baseline stereo matching based on local, affinely invariant regions," in *In Proceeding of British Machine Vision Conference*, pp. 412–425, 2000.

[55] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," *Computer Vision-ECCV 2006*, pp. 430–443, 2006.

[56] E. Rosten, R. Porter, and T. Drummond, "Faster and better: A machine learning approach to corner detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, pp. 105–119, 2010.

[57] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[58] H. Bay, T. Tuytelaars, and L. V. Gool, "SURF: Speeded up robust features," *EECV, LNCS:3951*, pp. 404–417, 2006.

[59] K. Mikolajczyk and C. Schmid, "Scale & affine invariant interest point detectors," *International Journal of Computer Vision*, vol. 60, no. 1, pp. 63–86, 2004.

[60] W. Förstner, T. Dickscheid, and F. Schindler, "Detecting interpretable and accurate scale-invariant keypoints," in *12th IEEE International Conference on Computer Vision (ICCV'09)*, 2009.

[61] W. Förstner, "A framework for low level feature extraction," in *Proceedings of the third European conference on Computer Vision (Vol. II)*, ECCV '94, pp. 383–394, 1994.

[62] J. Bigün, "A structure feature for some image processing applications based on spiral functions," *Comput. Vision Graph. Image Process.*, vol. 51, no. 2, pp. 166–194, 1990.

[63] T. Dickscheid, F. Schindler, and W. Förstner, "Coding images with local features," *International Journal of Computer Vision*, vol. 94, pp. 154–174, 2011.

[64] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary Robust Independent Elementary Features," in *European Conference on Computer Vision*, pp. 778–792, 2010.

[65] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski, "Orb: An efficient alternative to sift or surf," in *ICCV*, pp. 2564–2571, 2011.

[66] A. Davison, "Slam with a single camera," in *Concurrent Mapping and Localization*, 2002.

[67] P. Gemeiner, P. Einramhof, and M. Vincze, "Simultaneous motion and structure estimation by fusion of inertial and vision data," *International Journal of Robotics Research*, vol. 26, pp. 591–605, 2007.

[68] K. Briechle and U. D. Hanebeck, "Template Matching Using Fast Normalized Cross Correlation," in *Proceedings of SPIE: Optical Pattern Recognition XII*, vol. 4387, pp. 95–102, 2001.

[69] A. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "MonoSLAM: Real-time single camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, pp. 1–16, 2007.

[70] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *In VISAPP International Conference on Computer Vision Theory and Applications*, pp. 331–340, 2009.

[71] M. Fischler and R. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, no. 24, pp. 381–395, 1981.

[72] S. Choi, T. Kim, and W. Yu, "Performance Evaluation of RANSAC Family," in *BMVC*, British Machine Vision Association, 2009.

[73] A. Méler, M. Decrouez, and J. L. Crowley, "Betasac: A new conditional sampling for ransac.," in *BMVC*, pp. 1–11, British Machine Vision Association, 2010.

[74] M. Chandraker, C. Stock, and A. Pinz, "Real-time camera pose in a room," *Lecture Notes in Computer Science*, vol. 2626, pp. 98–110, 2003.

[75] P. Corke, J. Lobo, and J. Dias, "An introduction to inertial and visual sensing," *International Journal of Robotics Research*, vol. 28, pp. 519–535, 2007.

[76] R. Haralick, H. Joo, D. Lee, S. Zhuang, V. Vaidya, and M. Kim, "Pose estimation from corresponding point data," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 19, no. 6, pp. 1426–1446, 1989.

[77] R. Haralick, D. Lee, K. Ottenburg, and M. Nolle, "Analysis and solutions of the three point perspective pose estimation problem," in *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR '91., IEEE Computer Society Conference on*, pp. 592–598, 1991.

[78] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer, 2011.

[79] V. Lepetit and P. Fua, "Monocular model-based 3D tracking of rigid objects: A survey," *Foundations and Trends in Computer Graphics and Vision*, vol. 1, no. 1, pp. 1–89, 2005.

[80] B. Triggs, P. Mclauchlan, R. Hartley, and A. Fitzgibbon, "Bundle adjustment a modern synthesis," in *Vision Algorithms: Theory and Practice, LNCS*, pp. 298–375, Springer Verlag, 2000.

[81] C. Kelley, *Iterative Methods for Optimization*. Frontiers in Applied Mathematics, Society for Industrial and Applied Mathematics, 1999.

[82] M. I. A. Lourakis and A. Argyros, "Sba: A software package for generic sparse bundle adjustment," *ACM Trans. Math. Software*, vol. 36, no. 1, pp. 1–30, 2009.

[83] C. Lu, G. Hager, and E. Mjolsness, "Fast and globally convergent pose estimation from video images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 6, pp. 610–622, 2000.

[84] G. Schweighofer and A. Pinz, "Robust pose estimation from a planar target," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, pp. 2024–2030, 2006.

[85] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd, "Real time localization and 3d reconstruction," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. 363–370, 2006.

[86] J. Chen and A. Pinz, "Structure and motion by fusion of inertial and vision-based tracking," in *Austrian Association for Pattern Recognition*, vol. 179, pp. 75–62, 2004.

[87] J. Civera, A. Davison, and J. M. M. Montiel, "Interacting multiple model monocular slam," in *International Conference on Robotics and Automation*, pp. 3704–3709, 2008.

[88] F. Moreno-Noguer, V. Lepetit, and P. Fua, "Accurate non-iterative O(n) solution to the PnP problem," in *IEEE International Conference on Computer Vision*, 2007.

[89] V. Lepetit, F. Moreno-Noguer, and P. Fua, "EPnP: An Accurate O(n) Solution to the PnP Problem," *International Journal of Computer Vision*, vol. 81, no. 2, pp. 155–166, 2009.

[90] P. Smith, I. Reid, and A. Davison, "Real-time monocular slam with straight lines," in *British Machine Vision Conference*, vol. 1, pp. 17–26, 2006.

[91] G. Bleser, C. Wohlleber, M. Becker, and D. Stricker, "Fast and stable tracking for ar fusing video and inertial sensor data," in *International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2006.

[92] Z. Chen, "Bayesian Filtering: From Kalman Filters to Particle Filters and Beyond," tech. rep., McMaster University, 2003.

[93] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME Journal of Basic Engineering*, no. 82 (Series D), pp. 35–45, 1960.

[94] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2006.

[95] G. Welch and G. Bishop, "An introduction to the kalman filter," tech. rep., Chapel Hill, NC, USA, 1995.

[96] N. Gordon, D. Salmond, and A. F. M. Smith, "Novel approach to nonlinear/non-gaussian bayesian state estimation," *Radar and Signal Processing, IEE Proceedings F*, vol. 140, no. 2, pp. 107–113, 1993.

[97] R. Karlsson and F. Gustafsson, "Recursive bayesian estimation: bearings-only applications," *Radar, Sonar and Navigation, IEE Proceedings -*, vol. 152, no. 5, pp. 305–313, 2005.

[98] K. Park, D. Chung, H. Chung, and J. Lee, "Dead reckoning navigation of a mobile robot using an indirect kalman filter," in *Multisensor Fusion and Integration for Intelligent Systems, 1996. IEEE/SICE/RSJ International Conference on*, pp. 132–138, 1996.

[99] B. Widrow and S. Haykin, *Least-mean-square adaptive filters*. Adaptive and learning systems for signal processing, communications, and control, Wiley-Interscience, 2003.

[100] N. Metropolis and S. Ulam, "The monte carlo method," *Journal of the American Statistical Association*, vol. 44, no. 247, pp. 335–341, 1949.

[101] R. Van Der Merwe, A. Doucet, N. De Freitas, and E. Wan, "The unscented particle filter," *Advances in neural information processing systems*, pp. 584–590, 2001.

[102] M. Isard and A. Blake, "Condensation – conditional density propagation for visual tracking," *International Journal of Computer Vision*, vol. 29, no. 1, pp. 5–28, 1998.

[103] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "Fastslam: a factored solution to the simultaneous localization and mapping problem," in *Eighteenth national conference on Artificial intelligence*, pp. 593–598, 2002.

[104] R. Sim, P. Elinas, and J. Little, "A study of the rao-blackwellised particle filter for efficient and accurate vision-based slam," *International Journal of Computer Vision*, vol. 74, no. 3, pp. 303–318, 2007.

[105] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping:part i," *IEEE Robotics & Automation Magazine*, pp. 99–107, 2006.

[106] F. Bonin-Font, A. Ortiz, and G. Oliver, "Visual navigation for mobile robots: A survey," *Journal of Intelligent and Robotic Systems*, vol. 53, no. 3, pp. 263–296, 2008.

[107] N. Adluru and L. J. Latecki, "Contour grouping based on contour-skeleton duality," *International Journal Computer Vision*, vol. 83, pp. 12–29, 2009.

[108] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping:part ii," *IEEE Robotics & Automation Magazine*, pp. 108–117, 2006.

[109] J. Neira, J. Tardos, and J. Castellanos, "Linear time vehicle relocation in slam," in *IEEE International Conference on Robotics and Automation*, pp. 427–433, 2003.

[110] T. Bailey, *Mobile Robot Localisation and Mapping in Extensive Outdoor Environments*. PhD thesis, Australian Centre for Field Robotics, Department of Aerospace, Mechanical and Mechatronic Engineering The University of Sydney, 2002.

[111] A. Kozlov, B. Macdonald, and B. Wunsche, "Towards improving slam algorithm development using augmented reality," in *Australasian Conference on Robotics and Automation*, 2007.

[112] L. M. Paz, *Divide and Conquer: EKF SLAM in O(n)*. PhD thesis, Departamento de Informatica e Ingenieria de Sistemas, Universidad de Zaragoza, 2008.

[113] J. Neira and J. Tardos, "Data association in stochastic mapping using the joint compatibility test," *IEEE Transactions on Robotics and Automation*, pp. 890–897, 2001.

[114] J. Montiel and A. Davison, "A visual compass based on slam," in *IEEE International Conference on Robotics and Automation*, pp. 1917–1922, 2006.

[115] L. Clemente, A. Davison, I. Reid, J. Neira, and J. Tardos, "Mapping large loops with a single hand-held camera," in *Robotics: Science and Systems*, 2007.

[116] L. Paz, P. Jenfelt, J. Tardos, and J. Neira, "Ekf slam updates in o(n) with divide and conquer slam," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1107–1120, 2008.

[117] J. Claraco, *Contributions to Localization, Mapping and Navigation in Mobile Robotics*. PhD thesis, Universidad de Malaga, 2009.

[118] T. Lemaire, S. Lacroix, and J. Sola, "A practical 3d bearing-only slam algorithm," in *International Conference on Intelligent Robots and Systems*, 2005.

[119] T. Lemaire, C. Berger, I. Jung, and S. Lacroix, "Vision-based slam: Stereo and monocular approaches," *International Journal of Computer Vision*, vol. 74, no. 3, pp. 343–364, 2007.

[120] R. Smith, M. Self, and P. Cheeseman, "A stochastic map for uncertain spatial relationships," in *International Symposium on Robotics Research*, pp. 467–474, 1988.

[121] A. Brooks and B. T., "Hybridslam: Combining fastslam and ekf-slam for reliable mapping," *Algorithmic Foundation of Robotics VIII*, pp. 647–661, 2009.

[122] K. Konolige and M. Agrawal, "Frameslam: from bundle adjustment to realtime visual mappping," *International Journal of Robotics Research*, pp. 1066–1077, 2008.

[123] L. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, pp. 338–353, 1965.

[124] L. Zadeh, "Fuzzy logic and approximate reasoning," *Synthese*, vol. 30, pp. 407–428, 1975.

[125] E. Mamdani, "A fuzzy rule-based method of controlling dynamic processes," in *Decision and Control including the Symposium on Adaptive Processes, 1981 20th IEEE Conference on*, vol. 20, pp. 1098–1103, 1981.

[126] T. Ross, *Fuzzy Logic with Engineering Applications*. Wiley, 2009.

[127] Y. Yam, P. Baranyi, and C. Yang, "Reduction of fuzzy rule base via singular value decomposition," *Fuzzy Systems, IEEE Transactions on*, vol. 7, no. 2, pp. 120–132, 1999.

[128] P. Baranyi and A. Varkonyi-Koczy, "Adaptation of svd-based fuzzy reduction via minimal expansion," *Instrumentation and Measurement, IEEE Transactions on*, vol. 51, no. 2, pp. 222–226, 2002.

[129] A. Azadegan, L. Porobic, S. Ghazinoory, P. Samouei, and A. S. Kheirkhah, "Fuzzy logic in manufacturing: A review of literature and a specialized application," *International Journal of Production Economics*, vol. 132, no. 2, pp. 258–270, 2011.

[130] R. Isermann, "On fuzzy logic applications for automatic control, supervision, and fault diagnosis," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 28, no. 2, pp. 221–235, 1998.

[131] M. Haciomeroglu, R. Laycock, and A. M. Day, "Fuzzy logic controlled pedestrian groups in urban environments," *Motion In Games*, vol. 7660, pp. 326–337, 2012.

[132] M. Barrow, *Statistics for Economics, Accounting and Business Studies*. Prentice Hall, 2001.

[133] G. Privitera, *Statistics for the Behavioral Sciences*. SAGE Publications, 2011.

[134] R. Sprinthall, *Basic statistical analysis*. Allyn and Bacon, 2003.

[135] W. Hays, *Statistics*. Harcourt Brace College Publishers, 1994.

[136] D. A. Berry, "Logarithmic transformations in ANOVA," *Biometrics*, vol. 43, no. 2, pp. 439–456, 1987.

[137] H. O. Hartley, "The use of range in analysis of variance," *Biometrika*, vol. 37, pp. 271–280, 1950.

[138] E. S. Pearson and H. O. Hartley, *Biometrika Tables for Statisticians*. Cambridge University Press, 1972.

[139] D. B. Duncan, "Multiple range and multiple F tests," *Biometrics*, vol. 11, pp. 1–42, 1955.

[140] R. A. Fisher, *The Design of Experiments*. Edinburgh: Oliver and Boyd, 1935.

[141] A. Hilton and R. A. Armstrong, "Statnote 6: Post-hoc anova tests," *Microbiologist*, vol. 2006, no. September, pp. 34–36, 2006.

[142] W. Broll, I. Lindt, I. Herbst, J. Ohlenburg, A. Braun, and R. Wetzel, "Toward next-gen mobile ar games," *IEEE Computer Graphics and Applications*, pp. 40–48, 2008.

[143] R. Azuma, "A survey of augmented reality," *Presence:Teleoperators and Virtual Environments*, vol. 6, pp. 355–385, 1997.

[144] M. Ribo, P. Lang, H. Ganster, M. Brandner, C. Stock, and A. Pinz, "Hybrid tracking for outdoor augmented reality applications," *IEEE Computer Graphics and Applications*, pp. 54–63, 2002.

[145] U. Neumann and S. You, "Natural feature tracking for augmented reality," *IEEE Transactions on Multimedia*, vol. 1, no. 1, pp. 53–64, 1999.

[146] D. Schmalstieg, l. G. Schal, D. Wagner, I. Barakonyi, G. Reitmayr, J. Newman, and F. Ledermann, "Managing complex augmented reality models," *IEEE Computer Graphics and Applications*, pp. 48–57, 2007.

[147] J. Kim and H. Jun, "Vision-based location positioning using augmented reality for indoor navigation," *IEEE Transactions on Consumer Electronics*, vol. 54, pp. 954–962, 2008.

[148] W. Piekarski and B. H. Thomas, "Augmented reality working planes: a foundation for action and construction at a distance," in *International Symposium on Mixed and Augmented Reality*, 2004.

[149] P. D. Ritsos, *Architectures for untethered augmented reality using wearable computers*. PhD thesis, Department of Electronic systems Engineering, University of Essex, 2006.

[150] E. Klopfer and K. Squire, "Environmental detectives: The development of an augmented reality platform for environmental simulations," *Educational Technology Research and Development*, vol. 56, pp. 203–228, 2007.

[151] A. Behzadan, B. W. Timm, and V. R. Kamat, "General-purpose modular hardware and software framework for mobile outdoor augmented reality applications in engineering," *Advanced Engineering Informatics*, vol. 22, pp. 90–105, 2008.

[152] G. Schall, E. Mendez, E. Kruijff, E. Veas, S. Junghanns, B. Reitinger, and D. Schmalstieg, "Handheld augmented reality for underground infrastructure visualization," *Personal Ubiquitous Computing*, vol. 13, pp. 281–291, 2009.

[153] D. Mountain and F. Liarokapis, "Mixed reality (mr) interfaces for mobile information systems," *Aslib Proceedings: New Information Perspectives*, vol. 59, pp. 422–436, 2007.

[154] B. F. Goldiez, A. M. Ahmad, and P. A. Hancock, "Effects of augmented reality display settings on human wayfinding performance," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 37, pp. 839–845, 2007.

[155] A. C. Lopes and J. M. S. Dias, "Integration of geo-referenced data for visual simulation in location-based mobile computing," *Computer-Aided Civil and Infrastructure Engineering*, vol. 21, pp. 514–529, 2006.

[156] T. Ishikawa, K. Thangamai, M. Kourigi, A. P. Gee, W. Mayol-Cuevas, K. Jung, and T. Kurata, "In-situ 3d indoor modeler with a camera and self-contained sensors," *Lecture Notes in Computer Science*, vol. 5622, pp. 454–464, 2009.

[157] W. Piekarski, "3d modeling with the tinmith mobile outdoor augmented reality system," *IEEE Computer Graphics and Applications*, pp. 14–17, 2006.

[158] T. Roden, I. Parberry, and D. Ducrestc, "Toward mobile entertainment: A paradigm for narrative-based audio only games," *Science of Computer Programming*, vol. 67, pp. 76–90, 2007.

[159] N. Kanwal, E. Bostanci, and A. F. Clark, "Kinect aided navigation system for visually impaired people," in *Proceedings of the Workshop on Recognition and Action for Scene Understanding (REACTS 2013)*, 2013.

[160] D. A. Bowman, E. Kruijff, J. J. LaViola, and I. Poupyrev, *3D User interfaces: Theory and Practice*. Addison-Wesley, 2005.

[161] T. Arvanitis, A. Petrou, J. Knight, S. Savas, S. Sotiriou, M. Gargalakos, and E. Gialouri, "Human factors and qualitative pedagogical evaluation of a mobile augmented reality system for science education used by learners with physical disabilities," *Personal and Ubiquitous Computing*, vol. 13, pp. 243–250, 2009.

[162] K. Yehuda, T. Kvan, and J. Affleck, *New Heritage: New Media and Cultural Heritage*. Routledge Publishing, 2007.

[163] D. Stricker and T. Kettenbach, "Real-time and markerless vision-based tracking for outdoor augmented reality applications," in *International Symposium on Augmented Reality*, 2001.

[164] P. Dahne and J. Karigiannis, "Archeoguide: System architecture of a mobile outdoor augmented reality system," in *IEEE International Symposium on Mixed and Augmented Reality*, pp. 263–264, 2002.

[165] R. Chiara, V. Santo, U. Erra, and V. Scanarano, "Real positioning in virtual environments using game engines," in *Eurographics Italian Chapter Conference*, 2006.

[166] G. Papagiannakis, G. Singh, and N. Magnenat-Thalmann, "A survey of mobile and wireless technologies for augmented reality systems," *Computer Animation and Virtual Worlds*, vol. 19, pp. 3–22, 2008.

[167] B. Koyuncu and E. Bostanci, "Virtual reconstruction of an ancient site: Ephesus," in *Proceedings of the XIth Symposium on Mediterranean Archaeology*, pp. 233–236, Archaeopress, 2007.

[168] R. G. Laycock, D. Drinkwater, and A. M. Day, "Exploring cultural heritage sites through space and time," *ACM Journal on Computing and Cultural Heritage*, vol. 1, no. 2, 2008.

[169] G. Ryder, P. Flack, and A. M. Day, "A framework for real-time virtual crowds in cultural heritage environments," *International Symposium on Virtual Reality, Archaeology and Cultural Heritage*, 2012.

[170] F. Remondino, "Heritage recording and 3d modelling with photogrammetry and 3d scanning," *Remote Sensing*, vol. 3, no. 6, pp. 1104–1138, 2011.

[171] H. Richards-Rissetto, F. Remondino, G. Agugiaro, J. Robertsson, J. von-Schwerin, and G. Girardi, "Kinect and 3d gis in archaeology," in *International Conference on Virtual Systems and Multimedia*, pp. 331–337, 2012.

[172] A. Yilmaz, O. Jayed, and M. Shah, "Objects tracking a survey," *ACM Computing Surveys*, vol. 38, no. 4, 2006.

[173] U. Neumann, S. You, Y. Cho, J. Lee, and J. Park, "Augmented reality tracking in natural environments," in *International Symposium on Mixed Reality*, 1999.

[174] H. Kato, M. Billinghurst, I. Poupyrev, K. Imamoto, and K. Tachibana, "Virtual object manipulation on a table-top ar environment," in *Augmented Reality, 2000. (ISAR 2000). Proceedings. IEEE and ACM International Symposium on*, pp. 111–119, 2000.

[175] G. Welch, G. Bishop, L. Vicci, S. Brumback, K. Keller, and D. Colucci, "High-performance wide-area optical tracking: The hiball tracking system," *Presence: Teleoperators and Virtual Environments*, pp. 1–22, 2001.

[176] A. Golding and N. Lesh, "Indoor navigation using a diverse set of cheap, wearable sensors," in *Third International Symposium on Wearable Computers*, pp. 29–36, 1999.

[177] A. Cheok and Y. Li, "Ubiquitous interaction with positioning and navigation using a novel light sensor-based information transmission system," *Personal and Ubiquitous Computing*, vol. 12, pp. 445–458, 2008.

[178] D. Johnston and A. Clark, "A vision-based location system using fiducials," *Vision, Video, and Graphics*, pp. 1–8, 2003.

[179] D. Johnston, M. Fluery, A. Downton, and A. Clark, "Real-time positioning for augmented reality on a custom parallel machine," *Image and Vision Computing*, vol. 23, no. 3, pp. 271–286, 2005.

[180] K. Yeung, D. Johnston, and A. Clark, "A comparison of fiducial-based visual positioning systems," in *International Conference on Pattern Recognition*, vol. 4, 2006.

[181] K. Chia, A. Cheok, and S. Prince, "Online 6 dof augmented reality registration from natural features," in *International Symposium on Mixed and Augmented Reality*, pp. 305–313, 2002.

[182] B. Jiang, U. Neumann, and S. You, "A robust hybrid tracking system for outdoor augmented reality," in *IEEE Virtual Reality Conference*, 2004.

[183] Y. Park, V. Lepetit, and W. Woo, "Multiple 3d object tracking for augmented reality," in *IEEE International Symposium on Mixed and Augmented Reality*, pp. 117–120, 2008.

[184] H. Bekel, G. Heidemann, and H. Ritter, "Interactive image data labeling using self-organizing maps in an augmented reality scenario," *Neural Networks*, vol. 18, pp. 566–574, 2005.

[185] A. Adams, N. Gelfand, and K. Pulli, "Viewfinder alignment," *EUROGRAPHICS*, vol. 27, pp. 597–606, 2008.

[186] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg, "Pose tracking from natural features on mobile phones," in *IEEE International Symposium on Mixed and Augmented Reality*, pp. 125–134, 2008.

[187] D. Stichling, N. Esau, B. Kleinjohann, and L. Kleinjohann, "Real-time camera tracking for mobile devices: The visitrack system," *Real-Time Systems*, pp. 279–305, 2006.

[188] Y. Yamanaka, M. Kanbara, and N. Yokoya, "Localization of walking or running user with wearable 3d position sensor," in *17th International Conference on Artificial Reality and Telexistence 2007*, pp. 39–45, 2007.

[189] S. You, U. Neumann, and R. Azuma, "Orientation tracking for outdoor augmented reality registration," *IEEE Virtual Reality*, pp. 36–42, 1999.

[190] S. G. Chroust and M. Vincze, "Fusion of vision and inertial data for motion and structure estimation," *Journal of Robotic Systems*, vol. 21, no. 2, pp. 73–83, 2004.

[191] P. Lang, A. Kusej, A. Pinz, and G. Brasseur, "Inertial tracking for mobile augmented reality," in *IEEE Instruments and Measurement Technology Conference*, 2002.

[192] E. Foxlin and L. Naimark, "Vis-tracker: A wearable vision-inertial self-tracker," in *IEEE Virtual Reality*, pp. 199–206, 2003.

[193] S. Diverdi and T. Hollerer, "Heads up and camera down: A vision-based tracking modality for mobile mixed reality," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, pp. 500–512, 2008.

[194] N. Haala and J. Bohm, "A multi-sensor system for positioning in urban environments," *ISPRS Journal of Photogrammetry & Remote Sensing*, vol. 58, pp. 31–42, 2003.

[195] A. Sherstyuk, A. Treskunov, and B. Berg, "Fast geometry acquisition for mixed reality applicaitons using motion tracking," in *IEEE International Symposium on Mixed and Augmented Reality*, pp. 179–180, 2008.

[196] D. Chekhlov, A. Gee, A. Calway, and W. Mayol-Cuevas, "Ninja on a plane: Automatic discovery of physical planes for augmented reality using visual slam," in *IEEE International Symposium on Mixed and Augmented Reality*, 2007.

[197] "Ogre 3d game engine." `http://www.ogre3d.org/`, 2013. Last access: November, 2013.

[198] G. Bleser, *Towards Visual-Inertial SLAM for Mobile Augmented Reality*. PhD thesis, Fachbereich Informatik der Technischen Universitt Kaiserslautern, 2009.

[199] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 1–10, 2007.

[200] M. Pupilli, *Particle Filtering for Real-time Camera Localisation*. PhD thesis, Department of Computer Science, University of Bristol, 2006.

[201] G. Klein, *Visual Tracking for Augmented Reality*. PhD thesis, Department of Engineering, University of Cambridge, 2006.

[202] K. Konolige, M. Agrawal, R. Bolles, C. Cowan, M. Fischler, and B. Gerkey, "Outdoor mapping and navigation using stereo vision," *Experimental Robotics*, vol. 39, pp. 179–190, 2008.

[203] D. Koller, G. Klinker, E. Rose, D. Breen, R. Whitaker, and M. Tuceryan, "Real-time vision-based camera tracking for augmented reality applications," in *Symposium on Virtual Reality Software and Technology*, pp. 87–94, 1997.

[204] H. Rehbinder and B. Ghosh, "Pose estimation using line-based dynamic vision and inertial sensors," *IEEE Transactions on Automatic Control*, vol. 48, no. 2, pp. 186–199, 2003.

[205] M. Tomono, "Monocular slam using rao-blackwellised particle filter with exhaustive pose space search," in *IEEE International Conference on Robotics and Automation*, pp. 2421–2426, 2007.

[206] R. Sim, P. Elinas, and M. Griffin, "Vision-based slam using the rao-blackwellised particle filter," in *Workshop on Reasoning with Uncertainty in Robotics*, 2005.

[207] P. Pathirana, A. Bishop, A. Savkin, S. W. Ekanayake, and T. Black, "A method for stereo-vision-based tracking for robotic applications," *Robotica*, pp. 1–8, 2009.

[208] T. Vidal-Calleja, A. Davison, J. Andrade-Cetto, and D. Murray, "Active control for single camera slam," in *International Conference on Robotics and Automation*, 2006.

[209] M. Agrawal, K. Konolige, and R. Bolles, "Localization and mapping for autonomous navigation in outdoor terrains : A stereo vision approach," in *IEEE Workshop on Applications of Computer Vision*, pp. 7–12, 2007.

[210] D. Schleicher, L. M. Bergasa, M. Ocana, R. Barea, and M. Lopez, "Real-time hierarchical outdoor slam based on stereovision and gps fusion," *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, pp. 440–452, 2009.

[211] V. Bonato, J. A. Holanda, and E. Marques, "An embedded multi-camera system for simultaneous localization and mapping," *Lecture Notes in Computer Science*, pp. 109–114, 2006.

[212] E. Bostanci, N. Kanwal, S. Ehsan, and A. F. Clark, "Tracking methods for augmented reality," in *The 3rd International Conference on Machine Vision*, pp. 425–429, 2010.

[213] J. Civera, O. G. Grasa, A. J. Davison, and J. M. M. Montiel, "1-point RANSAC for EKF filtering: Application to real-time structure from motion and visual odometry," *Journal of Field Robotics*, pp. 609–631, 2010.

[214] H. Jin, S. Soatto, and A. J. Yezzi, "Multi-view stereo reconstruction of dense shape and complex appearance," *International Journal of Computer Vision*, vol. 63, pp. 175–189, 2005.

[215] N. A. Thacker, A. F. Clark, J. L. Barron, J. R. Beveridge, P. Courtney, W. R. Crum, V. Ramesh, and C. Clark, "Performance characterization in computer vision: A guide to best practices," *Computer Vision and Image Understanding*, vol. 109, pp. 305–334, 2008.

[216] K. Mikolajczyk and C. Schmid, "Scale and affine invariant interest point detectors," *International Journal of Computer Vision*, vol. 60, no. 1, pp. 63–86, 2004.

[217] S. Ehsan, N. Kanwal, A. F. Clark, and K. D. McDonald-Maier, "Improved repeatability measures for evaluating the performance of feature detectors," *Electronics Letters*, vol. 46, Jul 2010.

[218] J. Sola, *Towards Visual Localization, Mapping and Moving Objects Tracking by a Mobile Robot: a Geometric and Probabilistic Approach*. PhD thesis, Institut National Politechnique de Toulouse, 2007.

[219] M. Perdoch, J. Matas, and S. Obdrzalek, "Stable affine frames on isophotes," in *ICCV*, 2007.

[220] T. Dickscheid and W. Förstner, "Evaluating the suitability of feature detectors for automatic image orientation systems," in *ICVS 2009, LNCS:5815*, pp. 305–314, 2009.

[221] T. Tuytelaars, "Dense interest points," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'10)*, pp. 2281–2288, 2010.

[222] S. Ehsan, N. Kanwal, A. F. Clark, and K. McDonald-Maier, "Measuring the coverage of interest point detectors," in *Image Analysis and Recognition (ICIAR'11), LNCS:6753*, pp. 253–261, 2011.

[223] H. Aanæs, A. L. Dahl, and K. S. Pedersen, "Interesting interest points—A comparative study of interest point performance on a unique data set," *International Journal of Computer Vision*, pp. 1–18, 2011.

[224] M. J. Crawley, *The R Book*. Wiley-Blackwell, 2007.

[225] R. S. Bivand, E. Pebesma, and V. Gomez-Rubio, *Applied Spatial Analysis with R*. Springer, 2008.

[226] B. W. Silverman, "Density estimation for statistics and data analysis," *Monographs on Statistics and Applied Probability*, 1986.

[227] A. Baddeley, "Analysing spatial point patterns in R." Workshop Notes, 2008.

[228] J. J. Lennon, "Red-shifts and red herrings in geographical ecology," *Ecography*, vol. 23, pp. 101–113, 2000.

[229] S. Shekar and H. Xiong, *Encyclopedia of GIS*. Springer, 2008.

[230] I. E. Sutherland, "Sketch pad a man-machine graphical communication system," in *Proceedings of the SHARE design automation workshop*, (New York, NY, USA), pp. 329–346, ACM, 1964.

[231] F. Mufti, R. Mahony, and J. Heinzmann, "Robust estimation of planar surfaces using spatio-temporal RANSAC for applications in autonomous vehicle navigation," *Robotics and Autonomous Systems*, vol. 60, no. 1, pp. 16–28, 2012.

[232] P. Hough, "Method and Means for Recognizing Complex Patterns." U.S. Patent 3.069.654, 1962.

[233] P. M. Dixon, "Ripley's K function," *Encyclopedia of Environmetrics*, vol. 3, pp. 1796–1803, 2002.

[234] S. B. A. SB, S. J. Melly, B. N. Sanchez, A. Patel, S. Buka, and S. L. Gortmaker, "Clustering of fast-food restaurants around schools: a novel application of spatial statistics to the study of food environments," *American Journal of Public Health*, vol. 95, pp. 1575–1581, 2005.

[235] S. Spielman, "Appropriate use of the k function in urban environments," *American Journal of Public Health*, vol. 96, p. 205, Feb 2006.

[236] L. O. Martins, E. C. Silva, A. C. Silva, A. C. Paiva, and M. Gattass, "Classification of breast masses in mammogram images using Ripley's k function and support vector machine," in *Proceedings of the 5th International Conference on Machine Learning and Data Mining in Pattern Recognition*, pp. 784–794, Springer-Verlag, 2007.

[237] E. Bostanci, N. Kanwal, and A. F. Clark, "Feature coverage for better homography estimation: An application to image stitching," in *Proceedings of the IEEE International Conference on Systems, Signals And Image Processing*, Apr 2012.

[238] B. D. Ripley, "The second order analysis of stationary point processes," *Journal of Applied Probability*, vol. 13, pp. 255–266, 1976.

[239] B. D. Ripley, "Modelling spatial patterns," *Journal of the Royal Statistics Society*, vol. 39, pp. 172–212, 1977.

[240] R. Szeliski, "Image alignment and stitching: a tutorial," *Foundations and Trends in Computer Graphics and Vision*, vol. 2, no. 1, pp. 1–104, 2006.

[241] J. E. Davis, "Combining error ellipses." 2007.

[242] W. Smith, *Modern Lens Design*. Mcgraw-hill, 2004.

[243] D. C. Brown, "Decentering distortion of lenses," *Photogrammetric Engineering*, vol. 32, no. 3, pp. 444–462, 1966.

[244] D. Gu, P. Petkov, and M. Konstantinov, *Robust Control Design with MATLAB®*. Advanced Textbooks in Control and Signal Processing, Springer, 2005.

[245] Y. Tian and N. Sarkar, "Control of a mobile robot subject to wheel slip," *Journal of Intelligent & Robotic Systems*, pp. 1–15, 2013.

[246] S. Haykin, *Kalman Filtering and Neural Networks*. John Wiley & Sons, Inc., 2001.

[247] M. S. Grewal and A. P. Andrews, *Kalman Filtering: Theory and Practice Using MATLAB*. John Wiley & Sons, Inc., 2001.

[248] E. Eade, *Monocular Simultaneous Localisation and Mapping*. PhD thesis, University of Cambridge, Department of Engineering, 2008.

[249] J. Montiel, J. Civera, and A. Davison, "Unified inverse depth parametrization for monocular slam," in *Robotics:Systems and Science*, 2006.

[250] E. Eade and T. Drummond, "Monocular slam as a graph of coalesced observations," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pp. 1–8, 2007.

[251] D. Schleicher, L. M. Bergasa, M. Ocana, and E. Lopez, "Real-time hierarchical stereo visual SLAM in large-scale environments," *Robotics and Autonomous Systems*, vol. 58, no. 8, pp. 991–1002, 2010.

[252] H. C. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections," *Nature*, vol. 293, pp. 133–135, 1981.

[253] E. Royer, M. Lhuillier, D. M., and C. T., "Localization in urban environments: Monocular vision compared to a differential gps sensor," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 114–121, 2005.

[254] A. Geiger, J. Ziegler, and S. C., "Stereoscan: Dense 3d reconstruction in real-time," in *IEEE Intelligent Vehicles Symposium*, 2011.

[255] C. Fischer, P. T. Sukumar, and M. Hazas, "Tutorial: Implementing a pedestrian tracker using inertial sensors," *IEEE Pervasive Computing*, vol. 12, no. 2, pp. 17–27, 2013.

[256] J. Sola, *Towards Visual Localization, Mapping and Moving Objects Tracking by a Mobile Robot: a Geometric and Probabilistic Approach*. PhD thesis, lInstitut National Politechnique de Toulouse, 2007.

[257] B. Williams and I. Reid, "On combining visual slam and visual odometry," in *Proc. International Conference on Robotics and Automation*, 2010.

[258] R. I. Hartley, "In defense of the eight-point algorithm," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 6, pp. 580–593, 1997.

[259] Z. Zhang, "Motion and structure from two perspective views: From essential parameters to euclidean motion via fundamental matrix," *Journal of the Optical Society of America A*, pp. 2938–2950, 1997.

[260] B. K. P. Horn, "Recovering baseline and orientation from essential matrix," *J. Optical Society of America*, 1990.

[261] R. I. Hartley and P. Sturm, "Triangulation," *Computer Vision and Image Understanding*, vol. 68, no. 2, pp. 146–157, 1997.

[262] O. Rodrigues, "Des lois géométriques qui régissent les déplacements d'un systéme solide dans l'espace, et de la variation des coordonnées provenant de ces déplacements considérés indépendamment des causes qui peuvent les produire," *Journal de Mathématiques Pures et Appliquées*, pp. 380–440, 1840.

[263] M. Corporation, "Kinect for xbox360." `http://www.xbox.com/en-GB/Kinect`, 2010. Last access: November, 2013.

[264] G. Simon, A. Fitzgibbon, and A. Zisserman, "Markerless tracking using planar structures in the scene," in *International Symposium on Augmented Reality*, 2000.

[265] R. Wahl, M. Guthe, and R. Klein, "Identifying planes in point-clouds for efficient hybrid rendering," in *13th Pacific Conference on Computer Graphics and Applications*, 2005.

[266] R. Schnabel, R. Wahl, and R. Klein, "Efficient RANSAC for point-cloud shape detection," *Computer Graphics Forum*, vol. 26, no. 2, pp. 214–226, 2007.

[267] M. Ying Yang and W. Forstner, "Plane detection in point cloud data," tech. rep., Department of Photogrammetry,Institute of Geodesy and Geoinformation, University of Bonn, 2010.

[268] D. Dube and A. Zell, "Real-time plane extraction from depth images with the randomized hough transform," in *ICCV Workshops*, 2011.

[269] D. C. C. Tam and M. Fiala, "A real time augmented reality system using GPU acceleration," in *Computer and Robot Vision (CRV), 2012 Ninth Conference on*, pp. 101–108, 2012.

[270] M. Weinmann, S. Wursthorn, and B. Jutzi, "Semi-automatic image-based co-registration of range imaging data with different characteristics," in *Photogrammetric Image Analysis PIA11, LNCS:6952*, pp. 119–124, 2011.

[271] K. Khoshelham, "Accuracy analysis of kinect depth data," *GeoInformation Science*, vol. 38, no. 5/W12, 2010.

[272] P. J. Schneider and D. H. Eberly, *Geometric Tools for Computer Graphics*. San Francisco: Morgan Kaufmann, 2003.

[273] `http://openkinect.org/wiki/Imaging_Information/`. Last access: November, 2013.

[274] "OpenMP." `http://openmp.org/wp/`, 2012. Last access: November, 2013.

[275] M. Suess and C. Leopold, "Common mistakes in openmp and how to avoid them - A collection of best practices," in *International Workshop on OpenMP*, 2006.

[276] R. M. Haralick, C. Lee, K. Ottenberg, and N. M., "Review and analysis of solutions of the three point perspective pose estimation problem," *International Journal of Computer Vision*, vol. 13, no. 3, pp. 331–356, 1994.

[277] E. Gedraite and M. Hadad, "Investigation on the effect of a gaussian blur in image filtering and segmentation," in *ELMAR, 2011 Proceedings*, pp. 393–396, 2011.

[278] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698, 1986.

[279] D. Douglas and T. Peuker, "Algorithms for the reduction of the number of points required to represent a digitised line or its caricature," *The Canadian Cartographer*, vol. 10, pp. 112–122, 1973.

[280] OpenNI, "OpenNI library." `http://openni.org/`. Last access: November, 2013.

[281] I. PrimeSense, "Primesense nite algorithms 1.5," 2011.

[282] P. Read and M. Meyer, *Restoration of Motion Picture Film*. Butterworth-Heinemann series in conservation and museology, Elsevier Science, 2000.

[283] P. Groves, *Principles of GNSS, inertial, and multi-sensor integrated navigation systems*. GNSS technology and applications series, Artech House, 2008.

[284] E. Kaplan and C. Hegarty, *Understanding GPS: Principles and Applications*. Artech House mobile communications series, Artech House, 2005.

[285] L. Armesto, J. Tornero, and V. M., "Fast ego-motion estimation with multi-rate fusion of inertial and vision," *The International Journal of Robotics Research*, vol. 26, pp. 577–589, 2007.

[286] D. Tornqvist, T. B. Schon, R. Karlsson, and F. Gustafsson, "Particle filter slam with high dimensional vehicle model," *Journal of Intelligent Robot Systems*, vol. 55, pp. 249–266, 2009.

[287] T. Oskiper, S. Samarasekera, and R. Kumar, "Multi-sensor navigation algorithm using monocular camera, imu and gps for large scale augmented reality," in *Mixed and Augmented Reality (ISMAR), 2012 IEEE International Symposium on*, pp. 71–80, 2012.

[288] A. Almagbile, J. Wang, and W. Ding, "Evaluating the performances of adaptive kalman filter methods in GPS/INS integration," *Journal of Global Positioning Systems*, vol. 9, no. 1, pp. 33–40, 2010.

[289] C. Tseng, C. Chang, and D. Jwo, "Fuzzy adaptive interacting multiple model nonlinear filter for integrated navigation sensor fusion," *Sensors*, vol. 11, pp. 2090–2111, 2011.

[290] J. Kramer and A. Kandel, "On accurate localization and uncertain sensors," *International Journal of Intelligent Systems*, vol. 27, no. 5, pp. 429–456, 2012.

[291] L. Ojeda and J. Borenstein, "Flexnav: fuzzy logic expert rule-based position estimation for mobile robots on rugged terrain," in *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, vol. 1, pp. 317–322 vol.1, 2002.

[292] S. K. Hong, "Fuzzy logic based closed-loop strapdown attitude system for unmanned aerial vehicle (uav)," *Sensors and Actuators A: Physical*, vol. 107, no. 2, pp. 109 – 118, 2003.

[293] P. Torr, "Bayesian model estimation and selection for epipolar geometry and generic manifold fitting," *International Journal of Computer Vision*, vol. 50, no. 1, pp. 35–61, 2002.

[294] K. Kanatani, "Uncertainty modeling and model selection for geometric inference," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 10, pp. 1307–1319, 2004.

[295] K. Schindler and D. Suter, "Two-view multibody structure-and-motion with outliers through model selection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 28, no. 6, pp. 983–995, 2006.

[296] G. M. . Research, "U.s. satellite export policy report retains gps restrictions." `http://www.insidegnss.com/node/3035`. Last access: November, 2013.

[297] S. Gleason and D. Gebre-Egziabher, *Gnss Applications and Methods*. GNSS technology and applications series, Artech House, Incorporated, 2009.

[298] R. Bajaj, S. Ranaweera, and D. Agrawal, "Gps: location-tracking technology," *Computer*, vol. 35, no. 4, pp. 92–94, 2002.

[299] B. Ben-Moshe, E. Elkin, H. Levi, and A. Weissman, "Improving Accuracy of GNSS Devices in Urban Canyons," in *Proceedings of the 23rd Canadian Conference on Computational Geometry (CCCG)*, 2011.

[300] S. H. Stovall, "Basic inertial navigation," Technical report, Naval Air Warfare Center Weapons Division, 2008.

[301] A. Persson, "The coriolis effect: Four centuries of conflict between common sense and mathematics. part i: A history to 1883," *History of Meteorology*, vol. 2, pp. 1–24, 2005.

[302] R. Oboe, R. Antonello, E. Lasalandra, G. Spinola, and L. Prandi, "Control of a z-axis mems vibrational gyroscope," in *Advanced Motion Control, 2004. AMC '04. The 8th IEEE International Workshop on*, pp. 153–158, 2004.

[303] S. Zotov, A. Trusov, and A. Shkel, "Demonstration of a wide dynamic range angular rate sensor based on frequency modulation," in *Sensors, 2011 IEEE*, pp. 149–152, 2011.

[304] M. Andrejasic, "MEMS accelerometers," Seminar, University of Ljubljana, Faculty for Mathematics and Physics, Department of Physics, 2008.

[305] E. Nebot and H. Durrant-Whyte, "Initial calibration and alignment of low-cost inertial navigation units for land vehicle applications," *Journal of Robotic Systems*, vol. 16, no. 2, pp. 81–92, 1999.

[306] K. J. Walchko and A. C. Mason, "Inertial navigation," in *Florida Conference on Recent Advances in Robotics Inertial Navigation*, 2002.

[307] Phidgets, "1040 gps specifications." `http://www.phidgets.com/products.php?product_id=1040_0`. Last access: November, 2013.

[308] Phidgets, "1056 spatial specifications." `http://www.phidgets.com/products.php?product_id=1056_0`. Last access: November, 2013.

[309] National Imagery and Mapping Agency, "Department of Defense World Geodetic System 1984: its definition and relationships with local geodetic systems," tech. rep., National Imagery and Mapping Agency, 2000.

[310] J. Kuipers, *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace, and Virtual Reality*. Princeton paperbacks, Princeton University Press, 2002.

[311] A. Silberschatz, G. Gagne, and P. Galvin, *Operating System Concepts*. Wiley John + Sons, 2013.

[312] J. Seo, J. G. Lee, and C. Park, "Leverarm compensation for integrated navigation system of land vehicles," in *Control Applications, 2005. CCA 2005. Proceedings of 2005 IEEE Conference on*, pp. 523–528, 2005.

[313] L. Kleeman, "Understanding and applying kalman filtering." `http://www.ecse.monash.edu.au/centres/irrc/LKPubs/Kalman.PDF`. Last access: November, 2013.

[314] I. Reid, "Estimation ii." `http://www.robots.ox.ac.uk/~ian/Teaching/Estimation/LectureNotes2.pdf`. Last access: November, 2013.

[315] R. Hawkey and N. Futurelab, *Learning with Digital Technologies in Museums, Science Centres and Galleries*. NESTA Futurelab Series, Futurelab Education, 2004.

[316] T. Wolfenstetter, "Applications of augmented reality technology for archaeological purposes," tech. rep., Technische Universität München, 2007.

[317] P. Ritsos, D. J. Johnston, C. Clark, and A. F. Clark, "Engineering an augmented reality tour guide," in *IEEE, Eurowearable*, 2003.

[318] M. N. Thalmann, A. Foni, G. Papagiannakis, and C. N. Yazli, "Real Time Animation and Illumination in Ancient Roman Sites," *The International Journal of Virtual Reality, IPI Press*, vol. 6, no. 1, 2007.

[319] Z. Noh, M. S. Sunar, and Z. Pan, "A review on augmented reality for virtual heritage system," in *Proceedings of the 4th International Conference on E-Learning and Games: Learning by Playing. Game-based Education System Design and Development*, pp. 50–61, 2009.

[320] Autodesk, "3D Studio Max." `http://usa.autodesk.com/3ds-max/`. Last access: November, 2013.

[321] Autodesk, "AutoCAD." `http://usa.autodesk.com/autocad/`. Last access: November, 2013.

[322] P. Heckbert, "Survey of texture mapping," *Computer Graphics and Applications, IEEE*, vol. 6, no. 11, pp. 56–67, 1986.

[323] F. I. Apollonio, C. Corsi, M. Gaiani, and S. Baldissini, "An integrated 3D geodatabase for Palladio's work," *International Journal of Architectural Computing*, pp. 111–133, 2010.

[324] A. R. M. Eshaq, P. Avijit, S. Noraishah, and M. Nazri, "Techniques on heritage preservation using lighting computation virtual environment," in *Joining Languages, Cultures and Visions: Proceedings of the 13th International CAAD Futures Conference*, pp. 95–104, 2009.

[325] Q. H. He, Y. Zhai, and X. W. Li, "Texture baking techniques on construction of virtual reality interactive scenes," *Applied Mechanics and Materials*, vol. 55-57, pp. 478–483, 2011.

[326] D. Hearn and P. Baker, *Computer Graphics*. Prentice-Hall, 1986.

[327] Irrlicht, "Irrlicht 3d game engine." `http://irrlicht.sourceforge.net/`. Last access: November, 2013.

[328] B. Evers, C. Thoenes, and G. Kunstbibliothek (Berlin, *Architectural Theory: From the Renaissance to the Present : 89 Essays on 117 Treatises*. Midi Series, Taschen, 2003.

[329] R. Hartley, "Minimizing algebraic error in geometric estimation problems," in *Computer Vision, 1998. Sixth International Conference on*, pp. 469–476, 1998.

[330] H. Chen, D. Sun, and J. Yang, "Global localization of multirobot formations using ceiling vision slam strategy," *Mechatronics*, vol. 19, pp. 617–628, 2009.

[331] J. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Bradford, 1992.