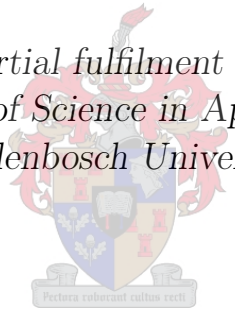
Enhancing Mobile Camera Pose Estimation Through the Inclusion of Sensors

by

Lloyd Haydn Hughes

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Applied Mathematics at Stellenbosch University



Department of Mathematical Sciences,
Stellenbosch University,
Private Bag X1, Matieland 7602, South Africa.

Supervisor: Dr Willie Brink

December 2014

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: December 2014

Abstract

Monocular structure from motion (SfM) is a widely researched problem, however many of the existing approaches prove to be too computationally expensive for use on mobile devices. In this thesis we investigate how inertial sensors can be used to increase the performance of SfM algorithms on mobile devices.

Making use of the low cost inertial sensors found on most mobile devices we design and implement an extended Kalman filter (EKF) to exploit their complementary nature, in order to produce an accurate estimate of the attitude of the device. We make use of a quaternion based system model in order to linearise the measurement stage of the EKF, thus reducing its computational complexity. We use this attitude estimate to enhance the feature tracking and camera localisation stages in our SfM pipeline.

In order to perform feature tracking we implement a hybrid tracking algorithm which makes use of Harris corners and an approximate nearest neighbour search to reduce the search space for possible correspondences. We increase the robustness of this approach by using inertial information to compensate for inter-frame camera rotation. We further develop an efficient bundle adjustment algorithm which only optimises the pose of the previous three key frames and the 3D map points common between at least two of these frames. We implement an optimisation based localisation algorithm which makes use of our EKF attitude estimate and the tracked features, in order to estimate the pose of the device relative to the 3D map points. This optimisation is performed in two steps, the first of which optimises only the translation and the second optimises the full pose. We integrate the aforementioned three sub-systems into an inertial assisted pose estimation pipeline.

We evaluate our algorithms with the use of datasets captured on the iPhone 5 in the presence of a Vicon motion capture system for ground truth data. We find that our EKF can estimate the device's attitude with an average dynamic accuracy of $\pm 5^{\circ}$. Furthermore, we find that the inclusion of sensors into the visual pose estimation pipeline can lead to improvements in terms of robustness and computational efficiency of the algorithms and are unlikely to negatively affect the accuracy of such a system. Even though we managed to reduce execution time dramatically, compared to typical existing techniques, our full system is found to still be too computationally expensive for real-time performance and currently runs at 3 frames per second, however the ever improving computational power of mobile devices and our described future work will lead to improved performance.

From this study we conclude that inertial sensors make a valuable addition into a visual pose estimation pipeline implemented on a mobile device.

Opsomming

Enkel-kamera struktuur-vanaf-beweging (*structure from motion*, SfM) is 'n bekende navorsingsprobleem, maar baie van die bestaande benaderings is te berekeningsintensief vir gebruik op mobiele toestelle. In hierdie tesis ondersoek ons hoe traagheidsensors gebruik kan word om die prestasie van SfM algoritmes op mobiele toestelle te verbeter.

Om van die lae-koste traagheidsensors wat op meeste mobiele toestelle gevind word gebruik te maak, ontwerp en implementeer ons 'n uitgebreide Kalman filter (*extended Kalman filter*, EKF) om hul komplementêre geaardhede te ontgin, en sodoende 'n akkurate skatting van die toestel se postuur te verkry. Ons maak van 'n kwaternioon-gebaseerde stelselmodel gebruik om die meetstadium van die EKF te lineariseer, en so die berekeningskompleksiteit te verminder. Hierdie afskatting van die toestel se postuur word gebruik om die fases van kenmerkvolging en kameralokalisering in ons SfM proses te verbeter.

Vir kenmerkvolging implementeer ons 'n hibriede volgingsalgoritme wat gebruik maak van Harris-hoekpunte en 'n benaderde naaste-buurpunt-soektog om die soekruimte vir moontlike ooreenstemmings te verklein. Ons verhoog die robuustheid van hierdie benadering, deur traagheidsinligting te gebruik om vir kamerarotasies tussen raampies te kompenseer. Verder ontwikkel ons 'n doeltreffende bondelaanpassingsalgoritme wat slegs optimeer oor die vorige drie sleutelraampies, en die 3D punte gemeenskaplik tussen minstens twee van hierdie raampies. Ons implementeer 'n optimeringsgebaseerde lokaliseringsalgoritme, wat gebruik maak van ons EKF se postuurafskatting en die gevolgde kenmerke, om die posisie en oriëntasie van die toestel relatief tot die 3D punte in die kaart af te skat. Die optimering word in twee stappe uitgevoer: eerstens net oor die kamera se translasie, en tweedens oor beide die translasie en rotasie. Ons integreer die bogenoemde drie sub-stelsels in 'n pyplyn vir postuurafskatting met behulp van traagheidsensors.

Ons evalueer ons algoritmes met die gebruik van datastelle wat met 'n iPhone 5 opgeneem is, terwyl dit in die teenwoordigheid van 'n Vicon bewegingsvasleggingstelsel was (vir die gelyktydige opneming van korrekte postuurdata). Ons vind dat die EKF die toestel se postuur kan afskat met 'n gemiddelde dinamiese akkuraatheid van $\pm 5°$. Verder vind ons dat die insluiting van sensors in die visuele postuurafskattingspyplyn kan lei tot verbeterings in terme van die robuustheid en berekeningsdoeltreffendheid van die algoritmes, en dat dit waarskynlik nie die akkuraatheid van so 'n stelsel negatief beïnvloed nie. Al het ons die uitvoertyd drasties verminder (in vergelyking met tipiese bestaande tegnieke) is ons volledige stelsel steeds te berekeningsintensief vir intydse verwerking op 'n mobiele toestel en hardloop tans teen 3 raampies per sekonde. Die voortdurende verbetering van mobiele toestelle se berekeningskrag en die toekomstige werk wat ons beskryf sal egter lei tot 'n verbetering in prestasie.

Uit hierdie studie kan ons aflei dat traagheidsensors 'n waardevolle toevoeging tot 'n visuele postuurafskattingspyplyn kan maak.

Acknowledgements

I would like to thank:

- the National Research Foundation of South Africa and MIH for funding this research,
- the MIH MediaLab for providing an enjoyable environment in which to work,
- my supervisor Willie Brink for his valuable input and guidance,
- my girlfriend Alyssa and my family for their love and support throughout this endeavour,
- and my fellow students in the MIH MediaLab who provided me with help, support and entertainment through my studies.

Contents

Chapter 1

Introduction

One of the main differences between the field of computer vision and other image processing fields is that computer vision strives to recover 3D structure of a scene from a collection of images [47]. A main focus of computer vision throughout the years has been to solve the structure from motion (SfM) problem, in which 3D structure is inferred from pairs of images captured by specialised, calibrated, stereo camera equipment. Structure from motion is similar to simultaneous localisation and mapping (SLAM) found in mobile robotics, however there are subtle differences. SLAM focuses on generating a map of an unknown environment while navigating and exploring the environment, and is performed on a platform which undergoes controlled motion. Thus in SLAM the odometery data can be used to assist the localisation and navigation pipelines. On the other hand, SfM focuses on recovering 3D structure from a series of 2D images captured while under the influence of uncontrolled motion. Therefore SfM systems do not have additional motion information to make use of.

Since the early 21st century, and with the advent of portable digital cameras and sophisticated mobiles devices, there has been a shift in focus to applying computer vision techniques to monocular image and video data captured by uncalibrated consumer devices. This problem has proven to be somewhat more difficult than the stereo case due to the larger number of degrees of freedom and thus larger uncertainties [10].

The turn of the century has also seen an increased focus in the amalgamation of computer vision and graphics techniques for use in what is now termed augmented reality (AR). Augmented reality is a way in which digital information is combined seamlessly into images or video of the real world. This allows information or objects to be placed into video in a way which makes them seem as if they are meant to be there. This amalgamation has been made possible via SfM and SLAM techniques to keep track of a camera's pose with respect to both the real world and a virtual scene, which are later combined into a seamless experience.

1.1 Motivation

In recent times there has been a push to see AR applications running in real-time in order to augment a user's live view of the world with useful information, while not distracting the user from what is going on in the world. Mobile devices provide a perfect platform for this form of see-through AR as their processing capabilities are expanding rapidly and most provide a high quality camera and screen which could be used to augment a user's view of the world.

In order to provide users with the best augmented experience it is vital to provide a jitter and lag free system. Reduced computational complexity in the vision processing algorithms is key to reducing lag from the system, while accurate pose estimation and more robust feature correspondence algorithms are the enablers for a jitter free augmented experience.

Furthermore, accurate pose estimation and a lag free system are the first steps to creating much more powerful mobile augmented reality systems, with 3D reconstruction capabilities for example, which can lead to having AR applications that are also aware of the structure of the world around the user and thus can augment digital information into the scene in a more intuitive manner [47, 20].

Apart from AR applications, accurate mobile pose estimation could also prove to be an enabling technology in many other fields. Such fields could include medicine, security, rapid prototyping and archaeology. For example, an archaeologist in the field could use his mobile device in order to rapidly generate a high resolution 3D model of an artifact which he has found, or create a full map of the archaeological site. The use of 3D modelling in archaeology is not a new concept and is commonly performed using expensive laser scanners [7]. Accurate pose estimation on mobile devices can reduce the cost and complexity of mapping and modelling, thus allowing for better preservation of archaeological sites and artifacts.

1.2 Problem Statement

While SfM and SLAM techniques have been the subject of research for a long time, that research has mainly focused on stereo vision, offline or soft real-time approaches. It is only in the last decade or so that SfM and SLAM research has begun to focus on the monocular vision problem and dense reconstruction techniques. For this reason many of the current solutions rely on adaptations of previous stereo techniques, instead of utilising the nature of video data to improve the performance and robustness of these feature tracking systems. While there are dense, monocular reconstruction systems which operate in real-time, they are far too computationally intensive to be implemented on a mobile device directly and are thus not able to provide a common framework for large-scale AR applications. Furthermore, the systems which have been implemented on mobile devices have not made use of the many sensors and improved hardware available to support processing [20, 10].

A major contributing factor to the poor performance of current SfM algorithms on mobile devices is the lack of processing power available. Mobile devices are specifically designed to maximise the trade-off between battery life and processing performance and even today's high-end mobile platforms, which incorporate sequential and parallel architectures, suffer from this reduced performance.

For these reasons, this study will investigate the use of additional sensors to support the computer vision pipeline and increase the robustness, accuracy and efficiency of the pose estimation stage. It will also look at how the coherent nature of video data can further be exploited to allow for faster and more robust feature tracking under the assumption of controlled user motion and rigid scenes.

1.2.1 Overview of Structure from Motion

Structure from motion (SfM) refers to the problem of simultaneously estimating both 3D scene structure and camera pose from a collection of 2D images or a video sequence. The main challenge faced in SfM type problems is how to take 2D point correspondences in two or more images and manipulate them in a mathematically sound manner in order to accurately recover the original 3D structure [47, 40].

Although many examples of SfM exist, and the problem has been solved in many different ways, they all build upon two well researched techniques: factorisation and filtering. Factorisation refers to solving for the camera projection matrices, and hence the rotation and translation matrices (referred to as the pose or extrinsic camera parameters) via mathematical relationships between the corresponding image points. These relationships are defined by a constraint called epipolar geometry [40, 14]. Filtering techniques are seen as a newer way of approaching the SfM problem, and use probability theory to predict the camera motion. Such a prediction is then corrected through the use of actual measurements [10]. Kalman filters and particle filters are often implemented to perform this type of filtered SfM.

Both filtering and factorisation have been used sufficiently to be classed as well understood, however both of these methods suffer from drawbacks. Factorisation methods tend to be computationally intensive but more accurate than filtering, when combined with global optimisation techniques such as bundle adjustment [50], while filtering is iterative and thus less computationally expensive (however it grows in complexity as more feature points are added to the 3D structure). Filtering is also inaccurate before convergence to the correct solution.

Most SfM techniques follow a similar structure in which corresponding points are determined between spatially separated images. These corresponding points are then used to obtain the extrinsic camera parameters, either via factorisation or filtering. The extrinsic camera parameters are then used to triangulate the corresponding points in order to obtain a sparse reconstruction (or map) of the image scene. The generic SfM pipeline can be seen in figure 1.1.

The feature tracking stage of the SfM pipeline can use either optical flow or feature detection and matching techniques. Both these techniques are well defined

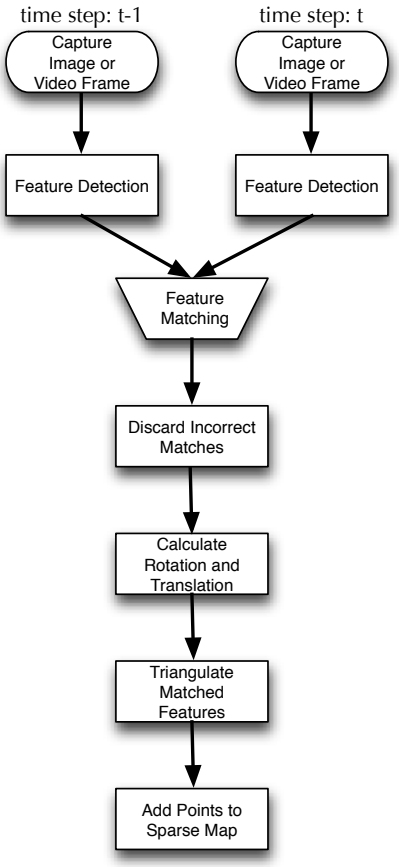


Figure 1.1: The generic structure from motion pipeline for sparse reconstruction.

with the main trade-off being between the speed of optical flow and the robustness of feature detection and matching.

The reconstruction from the generic SfM pipeline is only valid up to a scale factor, unless the size of an object in the scene is known a priori and is used to initialise the system.

In order for the structure to be correctly reconstructed, SfM techniques rely on the accuracy of the mathematical relationship between the cameras. Thus if we have an inaccurate estimate of the camera pose, we will obtain an inaccurate reconstruction.

1.2.2 Difficulties in Structure from Motion

While there has been much research and development in the field of SfM, the SfM problem as a whole remains incredibly difficult to solve in a robust and generic manner. There are a number of reasons which make SfM harder to solve than other problems in computer vision.

A major difficulty in SfM is the inter-dependency between the 3D positions of detected features and the camera pose. This means that the projected features on the image plane depend on the pose, but the camera pose can only be determined by knowing where these features are located in 3D. (This challenge is often referred to as a chicken and egg problem.)

Structure from motion systems also suffer from drift, due to the accumulation of estimation errors, noise and the fact that motion and structure are relative as no absolute position is known.

Another problem is encountered when ambiguous image data is captured. This refers to image regions that are hard to distinguish from each other based on features alone. Such ambiguous data is usually in the form of flat, mono-coloured and low texture regions such as walls and planar surfaces, or repeating patterns.

Previously optimisation techniques have been used to reduce drift and increase the accuracy of the estimated structure [50] and Monte Carlo techniques have been applied to improve the quality and efficiency of feature matching in the presence of ambiguity [32]. However, there is still no single solution which provides better results than others, or addresses both these issues in a coupled manner.

1.2.3 The Use of Monocular Vision

There are a lot of systems which use SfM, however it is only in recent times that these systems have begun to be implemented with a single camera. In the past most SfM and SLAM systems used calibrated, stereo camera setups which are composed of two cameras spaced at a known distance apart. Using such a setup allows for a reduced mathematical complexity as it makes single time step reconstruction possible.

Monocular (or single camera) SfM suffers from the problem that the camera motion is open to six degrees of freedom (DoF), that is three for translation and three for rotation, and the camera pose between successive frames needs to be calculated prior to any reconstruction being done. This problem is further complicated by the fact that many techniques rely on the system being initialised with known 3D points so that the camera pose can be calculated relative to the scene.

Monocular SfM has one obvious advantage over stereo SfM in that monocular SfM requires only one camera. This leads to a reduced implementation cost as well as the advantage that monocular SfM can be implemented on any device which has a single camera and sufficient processing power. This fact alone opens up the world of AR to most modern mobile devices.

1.2.4 The Use of Inertial Sensors

A common trend which has developed is the inclusion of inertial sensors on mobile platforms. The sensors which are included by most high-end device manufactures include gyroscopes, accelerometers and magnetometers.

Individually these sensors suffer from bias, drift and noise, and are thus not robust enough to be used independently. However, they can be combined using sensor fusion algorithms to provide higher accuracy estimates of attitude and translation [39, 48, 1].

1.2.5 The Use of Video Data

Until recently many SfM implementations relied solely on individual images or key frames extracted from video streams. While these approaches do have advantages it is theorised that using more video data can lead to improved robustness in the pose estimation pipeline.

Most of the existing approaches do not utilise every video frame. By doing so they reduce the computational load and thus allow for more expensive algorithms, such as bundle adjustment, to be used. Optimisation algorithms such as bundle adjustment are expensive to execute yet offer a more accurate pose estimate than factorisation based algorithms. Furthermore, optimisation based pose estimation also allows for zero motion and pure rotation to be solved, where factorisation based approaches require a significant baseline in order to solve for the pose and structure [33].

Another challenge encountered within the key frame approach is that the location and appearance of feature points in key frames can be vastly different and thus the SfM pipeline needs to rely on feature detection and matching as well as robust statistics, such as RANSAC, in order to limit the number of false positive feature matches [14].

By using almost every frame in a video sequence with constrained motion there is a guarantee that the appearance and location of the features are less likely to undergo extreme changes between frames. Furthermore, it can also be guaranteed that a large number of features should be visible between frames.

1.3 Related Work

Monocular SfM has advanced at a fairly rapid rate since its conception in 2002 when Pollefeys et al. created a system which generates a 3D reconstruction of a scene from uncalibrated video shot with a single camera [38]. Their system uses the factorisation approach and the generic SfM pipeline shown in figure 1.1, but adds an additional stage which rectifies two video frames using the computed camera parameters. These rectified frames are then processed using well known stereo techniques to generate a dense reconstruction. These dense reconstructions, created from two consecutive key frames, are then fused to create a final, dense, 3D reconstruction of the scene. Additional stages of bundle adjustment and key frame selection based on the disparity between the images are added to further improve the results.

While this system works well for a first implementation of monocular SfM, it does suffer a drawback that all the processing is done in an offline manner. Furthermore

it does not exploit the nature of video, or use the previously computed world map in order to make the system faster and more robust.

Since the original work of Pollefeys et al. on monocular SfM several improved systems, which take varying approaches to the monocular SfM problem, have been implemented [37, 52]. While these implementations produce good results, they still are computationally expensive and are thus performed in an offline manner. Furthermore these approaches rely heavily on the use of feature matching or user input instead of taking advantage of the structured nature of video data. The problem with offline processing is that it is not suitable for AR. If the camera moves to an unmapped area the AR system will not be able to augment information into the scene, and will thus provide a very poor user experience.

While many solutions are processed in an offline manner, there are a few which have become the state of the art in newer, real-time SfM and SLAM systems. Many of the contributions made to the field can be attributed to Davison and Newcombe [10, 34, 35] as well as to Klein and Murray [19, 20].

In 2007, Davison et al. proposed an approach to monocular SLAM which focuses on solving issues relating to the dynamic 3D motion of a single camera, processing efficiency and the use of commodity hardware [10]. The approach is now referred to as MonoSLAM. It makes use of an extended Kalman filter (EKF) and feature detection and matching to track the camera's position for every frame relative to the sparse 3D feature map which is simultaneously being created. The main difference between the system of Davison et al. and other implementations of SLAM at the time is in the way in which MonoSLAM handles feature detection and matching. The MonoSLAM system tracks only a few robust SIFT features in each frame and ensures that these features are spatially separated in an approximately uniform manner across the frame. Furthermore, the feature matching algorithm used in MonoSLAM does not find all the features in the next frame and tries to match them in a nearest neighbour approach, as was done before, but rather uses the motion prediction model from the EKF to try and predict which 3D feature points will be visible in the next frame and the locations of these points in the image plane. From this prediction a 2D correlation search in the predicted area is used to match the features. It is these techniques which provide the MonoSLAM system with the speed increase needed to operate in real-time.

While MonoSLAM works sufficiently well it has a problem in that the map needs to be manually initialised with at least three features in order for the system to work. This means that a user will need to know the position of three features with minimal uncertainty in order for the system to remain stable. Furthermore, the MonoSLAM system relies on an EKF which has a computational cost of $O(n^3)$, where n is the number of features in the map. This means that the larger the map becomes the slower the response time of the MonoSLAM system will be [46].

In 2007, Klein and Murray approached the SLAM problem in a novel manner and developed a system which removes the need for prior scene knowledge and scene map initialisation while maintaining real-time performance [19]. Their system manages

to obtain these improvements by approaching the tracking and mapping problem in a parallel rather than sequential manner. This allows for more computationally intensive batch optimisation techniques, not normally associated with real-time performance, to be used. The parallel tracking and mapping (PTAM) system proposed by Klein and Murray runs a tracking procedure on every frame in the video. This procedure re-projects the 3D image features from the current known map onto the image frame using the prior pose estimate. This estimate is then updated according to where the features actually occur in the current frame. The mapping thread performs two main operations, namely a key frame insertion step and a bundle adjustment step. The key frame insertion step adds new features to the map based on a key frame, which is selected based on two criteria: the number of frames since the last key frame needs to exceed 20 frames, and the camera needs to exceed a minimum distance from a known keypoint in the map. The bundle adjustment operation runs in-between the key frame insertion steps, and optimises the map on both a local and global scale.

The PTAM system has proven itself to be the new state-of-the-art system for use in AR and camera tracking applications and has been used in both its native form and as part of larger systems [20, 34, 35]. As with MonoSLAM the PTAM system also requires an initialisation step, however it is simpler than the MonoSLAM initialisation as it only requires the user to make a purely translational motion with a significant baseline as the first motion in the sequence [19].

Klein and Murray's PTAM has also been implemented on a mobile device. Albeit slower and less robust than the full implementation on a computer it still manages to perform scene augmentation at a rate of 15Hz [20]. Klein and Murray also mentioned that the narrow field of view of the iPhone 3, limited processing power, and the 15Hz rolling shutter camera impose the biggest limiting factors in implementing PTAM on a mobile device [20]. Although these limitations have now been reduced due to dual core CPUs and high quality cameras available on current mobile technology, the system is still not robust enough to provide a high quality augmented reality experience.

Feature tracking and mapping systems such as MonoSLAM and PTAM are the enabling technologies behind dense 3D SfM systems, as they provide the critical camera motion estimates which allow for points to be triangulated to create a dense 3D model of the scene. In 2011, Newcombe and Davison utilised the power and scalability of the PTAM system to work as the backend in their monocular, live, dense reconstruction system which is known to be one of the first real-time, dense, 3D reconstruction systems successfully implemented [34].

With the successful implementation of their live dense reconstruction system, Newcombe and Davison used what they learnt to implement a new system which generates a far more accurate dense map, and is also robust under fast motion [35]. This system, unlike previous systems, utilises every-pixel tracking techniques instead of feature detection and matching techniques as used by MonoSLAM, PTAM and their earlier live dense reconstruction system. Their new dense tracking and mapping

(DTAM) approach utilises the concurrent model borrowed from the PTAM system to accurately track the camera motion between frames in one thread and to generate a map with every key frame, and optimise the map using a global optimisation function [35].

The DTAM system manages to obtain superior tracking due to its 6DoF whole image alignment against the dense model, as well as its use of commodity GPU hardware for processing hundreds of video frames and running the optimisation functions. Newcombe and Davison also managed to show the usefulness of the dense model and tracking system for real-time, physics enabled AR systems [35].

Although all the above mentioned systems have proven to successfully overcome the challenges of monocular SfM and SLAM, only the DTAM system has taken full advantage of the nature of video and the ability to compute in a massively parallel fashion on commodity GPU hardware. It is these two features that have lead to the DTAM system's overall superior performance. However, all these systems have proven to be too computationally complex to run robustly on the reduced hardware found in mobile devices. Furthermore, none of the aforementioned systems rely on the use of any additional sensors to support the vision algorithms.

In order to improve the performance of AR applications on mobile devices there has been a move towards incorporating inertial information to assist the computer vision algorithms. Most of today's high-end mobile platforms come with a wide array of sensors. These include gyroscopes, magnetometers, accelerometers and global positioning systems (GPS).

The most popular sensor to integrate into the vision pipeline is the GPS, as it allows an easy way to obtain globally accurate position and translation data. While this works well for large scale, outdoor applications it is not applicable to small scene, indoor applications such as those which are the focus of this thesis, and thus will not be discussed further.

In 2014, Tanskanen et al. developed a complete on-device 3D reconstruction pipeline which generates accurate, up-to-scale 3D models of objects using a high-end mobile device [48]. In contrast to the previously described systems, their system utilises the on-board sensors to assist the tracking and mapping process. This adds robustness to the system and allows the entire pipeline to be more resilient to fast motions. Furthermore, they use the accelerometer to help estimate the metric scale of the object, thus allowing for a full correctly scaled reconstruction to be made. Apart from these novel contributions the live metric 3D reconstruction system, as it is entitled, also makes use of efficient and accurate dense stereo matching in order to reduce the processing time of each key frame.

The system of Tanskanen et al. is the first of its kind to allow full 3D reconstruction to take place completely on a mobile device in real-time. Their system works by using the inertial sensors and an EKF to estimate the pose of the device relative to the world. This pose is then used to estimate the positions of the sparse map's feature points in the current frame, and a patch based method is used to match these re-projected map points to extracted feature points within a certain

radius. The rest of the pose estimation and key frame extraction system follows a similar approach to PTAM [19] with the additional criterion that key frames are also extracted when it is detected that the device motion has stopped after salient motion. This additional criterion reduces the chance of motion blur affecting the key frame. From here the key frames are used for the creation of a dense 3D model.

Apart from the recent developments made by Tanskanen et al. there have been a few other attempts to use inertial sensors to assist vision processing on mobile devices. Most of these attempts, however, do not strongly couple the inertial information into the whole pipeline and generally use only a small amount of inertial data to assist a specific task in the vision processing pipeline. Examples of these include the use of inertial data to assist with feature description and matching [22], feature tracking [18], and to reduce the effects of rolling shutter distortion [26].

1.4 Our Approach

The main focus of this thesis is to investigate how the inclusion of sensor information can enhance pose estimation on mobile devices. We utilise a PTAM-like approach for the SfM pipeline and extend this by incorporating pose information from sensor fusion algorithms to support the vision processing. As with PTAM we try to estimate pose at every good frame in the video sequence. The most severe drawback of this is that the overall computational load of the system is increased. We therefore offload as much processing as possible to the mobile device's GPU as well as utilise the sensor information to reduce the computational complexity of the problem, similar to the work of Tanskanen et al. [48].

We begin by initialising the system's map with two user selected key frames, feature matching and the 8-point algorithm. This map is then refined in a separate thread using bundle adjustment. The full bundle adjustment leads to globally optimised map points and camera positions. A similar approach is followed in both PTAM [19] and live metric reconstruction [48].

We then track map points between subsequent video frames using a hybrid patch based tracking scheme. To improve efficiency we use inertial information from the gyroscopes to approximate the relative change in attitude of the device, and then use this information to support the patch tracking process and pick new key frames when the appearances of patches change significantly.

Next we use a lightweight bundle adjustment approach to estimate the complete pose of the camera relative to the sparse world map. We use an EKF to estimate the absolute attitude of the device and then use this estimate to reduce the complexity of the iterative optimisation. If the current frame is deemed to be of poor quality it is discarded and the process continues onto the next frame.

If a current good frame's pose has changed significantly from the previous key frame it is marked as a key frame and used to update the sparse world map and the appearance models of the features being tracked. Once the number of tracked features falls below a specified threshold, more features are extracted from the current

key frame and the entire process repeats. Our proposed system is summarised in figure 1.2.

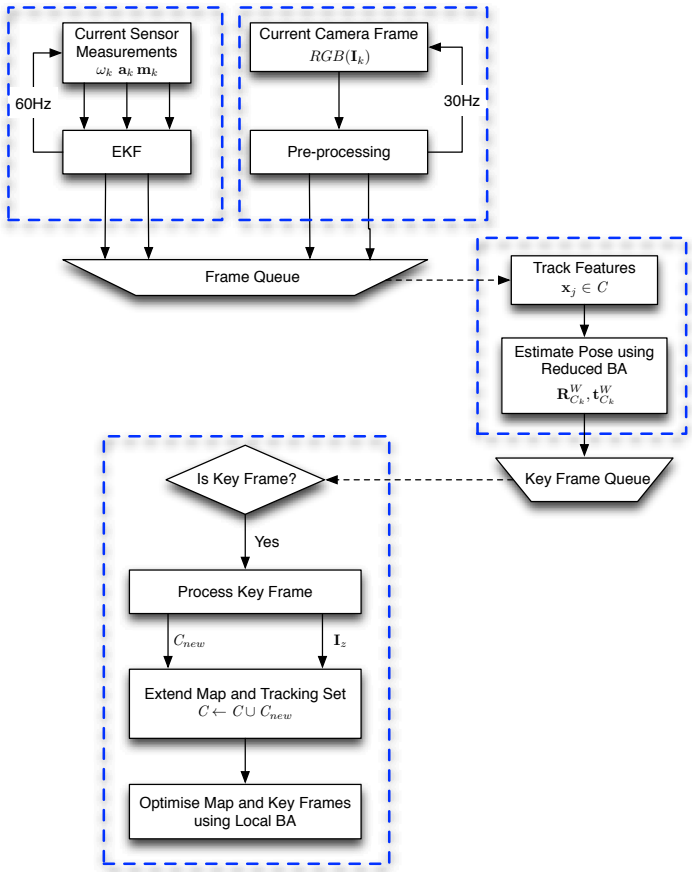


Figure 1.2: Our pose estimation pipeline. The arrows dictate the direction of information flow, dashed lines represent data being fetched from a queue and the blue boxes indicate the threads for the various tasks.

Our system is implemented entirely on the iPhone 5, which was deemed to be the most suitable and advanced device on the market at the commencement of this study. Furthermore, we use the GPU on the phone to assist with processing as much as possible. Running as many algorithms as we can on the GPU reduces the load on the CPU and allows us to run more complex optimisation algorithms in other threads. We sample the on-board sensors at 60Hz and the camera at 30Hz. The time between each of these samples must be used to update the Kalman filter states, perform localisation and update the map.

1.5 Contributions

This section lists the main contributions which resulted from the research done in this thesis. For aiding the development and evaluation of pose estimation algorithms on a mobile device we

- develop an application to synchronously capture inertial and video data from the sensors on the iPhone;
- create high quality datasets for use in designing and developing pose estimation algorithms for mobile devices. These datasets consist of inertial sensor data, video data and high accuracy ground truth data derived from a Vicon motion capture system.

For inertially aided pose estimation on a mobile device we

- present a discontinuity correction filter in an attempt to rectify factory miscalibration of the magnetometer;
- develop a hybrid feature tracking algorithm which combines feature detection and motion estimation, and argue why the approach is highly suitable for use on mobile devices;
- present a simplified formulation of bundle adjustment which reduces the size of the optimisation search space, and demonstrate that proper initialisation with inertial pose estimates can lead to increased robustness and decreased execution time.

We also identify challenges which need to be addressed in order to increase the feasibility and robustness of pose estimation on mobile devices.

1.6 Thesis Overview

This thesis starts with a description of the common inertial sensors found in modern mobile devices as well as a discussion on their noise characteristics and an overview of calibration procedures in chapter 2. As these sensors are complementary we discuss sensor fusion algorithms and describe an efficient sensor fusion algorithm which we refer back to throughout the rest of the thesis.

In chapter 3 we describe two key approaches to determining feature correspondence in sequential video frames. Using this information we propose a hybrid approach which makes use of feature detection, tracking techniques and inertial sensor information in order to enable fast and robust feature tracking on mobile devices.

After the definitions of feature correspondence and inertial pose estimation we discuss how this information can be used to recover 3D structure and camera pose

in chapter 4. We then go on to describe an alternative formulation of bundle adjustment which reduces the computational complexity of the algorithm. We further describe how inertial sensors can be used to assist in visual pose estimation and improve the robustness and speed of these algorithms.

Next we describe how our three main sub-systems can be integrated in order to provide an efficient mobile pose estimation system in chapter 5. This system can then be integrated into an augmented reality pipeline or 3D reconstruction system to provide an improved user experience.

We present our experimental results in chapter 6 and finally draw conclusions and discuss future work which can be undertaken in chapter 7.

Chapter 2

Inertial Pose Estimation

The iPhone 5 provides an estimate of its attitude from the CoreMotion library, however there is very little information on the accuracy and implementation of the filter used to determine this attitude. During basic testing it was determined that this attitude appears to have a worst case accuracy of $\pm 15°$, which is significant when dealing with a camera that has such a small field of view. For this reason we make use of the three available inertial sensors in order to compute the absolute and relative rotation between consecutive camera frames.

We utilise a filter based method, specifically an extended Kalman filter (EKF), to compute the absolute pose of the phone relative to the world coordinate system which coincides with the first camera frame in our system. The relative pose is obtained through a simple numerical integration as described in section 3.3.2. Unlike many other SfM systems which combine visual and inertial data, we do not incorporate visual data into our Kalman filter. The reason for this is to fix the computational complexity of the Kalman filter as well as to prevent bad visual data from negatively affecting our pose estimates. Kalman filters tend to be very sensitive to data that falls outside of their assumptions. Incorrect image feature matches can cause a Kalman filter to diverge away from the correct estimate which will require the filter to be re-initialised. By removing the need for visual data in the Kalman filter, we also reduce the chance of the filter diverging under uncontrolled motion.

In this chapter we describe our unique formulation of the extended Kalman filter as well as the pre-processing steps required in order to obtain an accurate pose from the low cost microelectromechanical system (MEMS) sensors found in mobile devices. Our EKF is based upon the work of Von Marcard [54] and Marins et al. [30], and makes use of quaternions in place of the general Euler representation for rotations used in many other systems. A full derivation and description of Kalman filtering is beyond the scope of this thesis and the reader is referred to Welsh and Bishop [55] for this.

2.1 Attitude Representation

In this section we describe the coordinate systems required. Furthermore, we describe three approaches to representing orientation in space, along with some advantages and disadvantages of each.

2.1.1 Inertial Coordinate System

In order to process inertial measurements we first need to define two coordinate systems, namely the body and the world coordinate system. We also need to define a mapping between these coordinate systems as well as a mapping between the inertial and camera coordinate systems, such that we can use inertial measurements in the camera coordinate system without the need for further processing.

The body coordinate system (or frame of reference) refers to the system of axes used by the sensors on the phone. This coordinate system is fixed to the device and thus moves as the device moves. The body coordinate system, represented with a subscript B, is a right-handed system of axes with the y-axis pointing up the screen, the x-axis pointing to the right and the z-axis facing the user. Refer to figure 2.1.

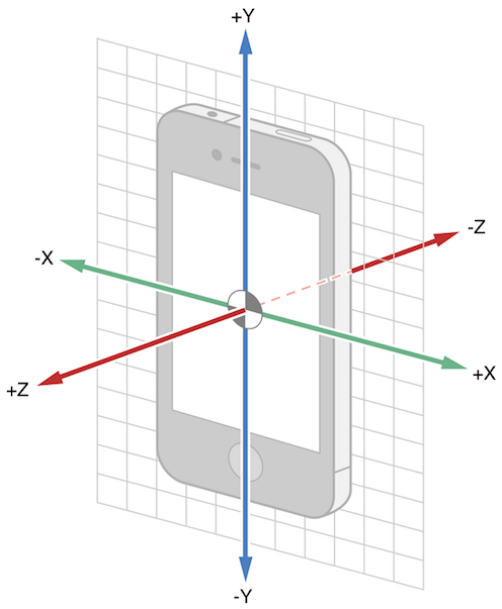


Figure 2.1: The body reference frame on the iPhone 5. (Image source: Apple developer documentation.)

In order to simplify the process and ensure our camera and inertial sensors are defined in the same world coordinate system, we use the first camera frame and the sensor data at this point to define our world coordinate system. This means that

everything in our system is relative to the first frame rather than relative to some global reference.

The inertial body frame of reference is related to the world coordinate system by a rotation $\mathbf{R}_B^W$ and a translation $\mathbf{t}_B^W$. We will use the EKF to estimate the rotation $\mathbf{R}_B^W$, but the inertial sensors are not accurate enough for estimating the translation. For this reason we will use the visual sensor data from the camera. The camera information will be used to estimate the rotation $\mathbf{R}_C^W$ and the translation $\mathbf{t}_C^W$, which relate the camera coordinate system with the world coordinate system. As the camera and inertial sensors are mounted on the same circuit board, and thus conform to the laws of rigid body motion, we know that the rotation experienced by the inertial sensors will be the same as that experienced by the camera. Thus we make use of the inertial rotation to help increase the robustness and reduce the computational complexity of the feature tracking and visual pose estimation stages, as discussed in chapters 3 and 4.

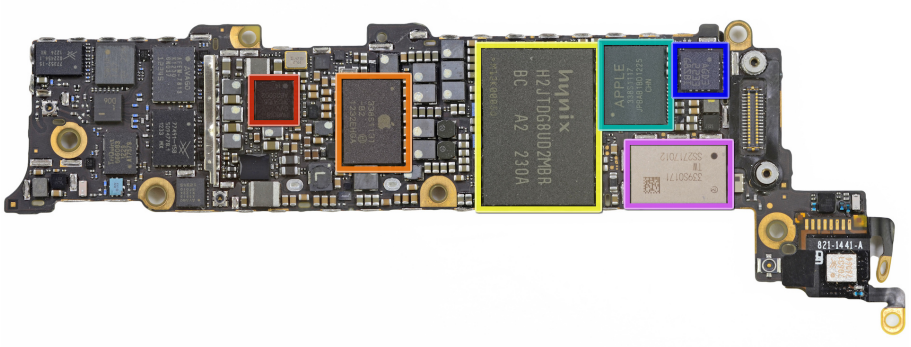


Figure 2.2: The internal circuitry of the iPhone 5. The position of the gyroscope is highlighted by the blue box, and the camera connector can be seen to the right of this chip. (Image source: iFixit, www.ifixit.com.)

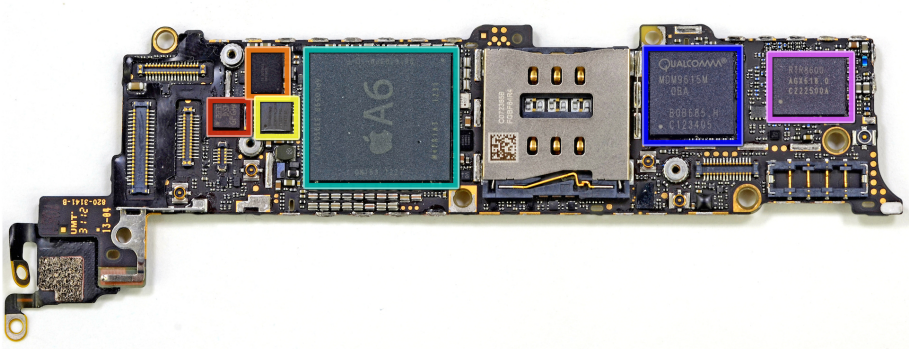


Figure 2.3: The reverse side of the circuit board in figure 2.2. The position of the accelerometer is highlighted by the red box. (Image source: iFixit, www.ifixit.com.)

As the visual and inertial sensors are not mounted in exactly the same place on the circuit board there is a translational offset between them, as can be seen in figures 2.2 and 2.3. This, however, has little effect as we only use the inertial sensor information to estimate rotation. Also, the inertial sensors in the iPhone 5 are placed in such a way that their orientation matches with the coordinate system of the camera. Although there is likely a small rotational misalignment between the sensors and the camera, it is assumed to be small enough to have a negligible effect in our system.

2.1.2 Euler Angles

The Euler angle representation of orientation makes use of three successive rotations around separate axes in order to describe a rotation in $\mathbb{R}^3$. Using this approach there are twelve unique ways to combine these three rotations in order to describe the final orientation in a right-handed system [11].

In this section we describe the yaw-pitch-roll (ZYX) convention, which represents the rotation by an angle ψ around the z-axis, followed by a rotation by θ about the y-axis and finally a rotation by ϕ about the x-axis. We can now describe an operation which maps these Euler angles into a corresponding rotation matrix:

$$\begin{aligned}\mathbf{R}_B^W &= \mathbf{R}_x(\phi)\mathbf{R}_y(\theta)\mathbf{R}_z(\psi) \\ &= \begin{bmatrix}1 & 0 & 0\\ 0 & \cos(\phi) & \sin(\phi)\\ 0 & -\sin(\phi) & \cos(\phi)\end{bmatrix}\begin{bmatrix}\cos(\theta) & 0 & -\sin(\theta)\\ 0 & 1 & 0\\ \sin(\theta) & 0 & \cos(\theta)\end{bmatrix}\begin{bmatrix}\cos(\psi) & \sin(\psi) & 0\\ -\sin(\psi) & \cos(\psi) & 0\\ 0 & 0 & 1\end{bmatrix} \\ &= \begin{bmatrix}C_\theta C_\psi & C_\theta S_\psi & -S_\theta\\ -C_\phi S_\psi + S_\phi S_\theta C_\psi & C_\phi C_\psi + S_\phi S_\theta S_\psi & S_\phi C_\theta\\ S_\phi S_\psi + C_\phi S_\theta C_\psi & -S_\phi C_\psi + C_\phi S_\theta S_\psi & C_\phi C_\theta\end{bmatrix},\end{aligned} \tag{2.1.1}$$

where C_θ is the cosine of the angle θ and S_θ is the sine of the angle θ.

We can use the rotation matrix in equation 2.1.1 to rotate a vector $\mathbf{x}_B$ from the body frame to the world frame of reference:

$$\mathbf{x}_W = \mathbf{R}_B^W \mathbf{x}_B. \tag{2.1.2}$$

Euler angles provide an intuitive means of representing rotations in $\mathbb{R}^3$, however they suffer from a singularity at pitch angle $\theta = \pm\frac{\pi}{2}$. At this angle the yaw and roll axes become parallel and the orientation no longer has a unique Euler angle representation. This singularity is referred to as gimbal lock, and is one of the main reasons Euler angles are not often used to parametrise rotations in systems that can experience the full range of rotations [11].

In the case of our system we cannot bound the rotation to a specific range and thus Euler angles are not sufficient for our purposes.

2.1.3 Direction Cosine Matrix

The direction cosine matrix (DCM) represents orientation between two coordinate systems by relating their basis vectors. In general, the basis vectors can be expressed as unit vectors pointing in the direction of the x-, y- and z-axes of a Cartesian coordinate system. Using this formulation we can write any vector $\mathbf{a}$ as a linear combination of basis vectors:

$$\mathbf{a} = a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + a_3\mathbf{e}_3 = a_1 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + a_2 \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + a_3 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}. \tag{2.1.3}$$

We can also express vector $\mathbf{a}$ as a linear combination of a different set of basis vectors:

$$\mathbf{a} = a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + a_3\mathbf{e}_3 = b_1\mathbf{n}_1 + b_2\mathbf{n}_2 + b_3\mathbf{n}_3. \tag{2.1.4}$$

Equation 2.1.4 can be rewritten in matrix form as

$$\mathbf{a} = \mathbf{C}_n^e\mathbf{b} \tag{2.1.5}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}, \tag{2.1.6}$$

where c_{nk} represents the cosine of the angle between the vectors $\mathbf{e}_n$ and $\mathbf{n}_k$, hence the name direction cosine matrix.

The DCM $\mathbf{C}_n^e$ represents a rotation from one coordinate system to another. Using this formulation we can see that we can express the rotation from the body to the world frame by the DCM $\mathbf{C}_B^W$, such that $\mathbf{C}_B^W = \mathbf{R}_B^W$. Thus the DCM can also be thought of as the matrix formed from Euler angles, such as the one in equation 2.1.1.

While the DCM is easy to formulate and does not suffer from singularities it is not intuitive to think of a rotation as a DCM. Furthermore, the DCM is hard to optimise and estimate as it contains nine parameters that are not all independent, which need to be estimated in such a manner that the matrix remains orthogonal. This is not impossible, but does require a fair amount of additional computation and thus was not deemed suitable for use in our system.

2.1.4 Quaternions

Quaternions parameterise rotations in $\mathbb{R}^3$ using an extension of the complex number system, and consist of an imaginary 3-vector $\mathbf{v}$ and a scalar w which together can be represented using a vector in $\mathbb{R}^4$. We generally make use of the notation

$$\mathbf{q} = (\mathbf{v}, w), \tag{2.1.7}$$

where $\mathbf{q}$ is a quaternion. Alternatively we may write $xi + yj + zk + w$ to denote the same quaternion, with x, y and z the components of $\mathbf{v}$. In this way $\mathbf{v}$ is made

up from three complex elements denoted as i, j and k, which have the following properties:

$$i^2 = j^2 = k^2 = ijk = -1, \tag{2.1.8}$$

$$\begin{aligned} ij = k,\ ji = -k, \\ jk = i,\ kj = -i, \\ ki = j,\ ik = -j. \end{aligned} \tag{2.1.9}$$

By interpreting the complex elements i, j and k as the basis vectors of a 3-dimensional Cartesian coordinate system we can intuitively think of a quaternion as defining an axis of rotation (the complex vector) and an angle of rotation (represented by the scalar w). Although this is not completely accurate it is by far the most intuitive interpretation of quaternions.

Since quaternions can be thought of as 4-dimensional vectors it may be postulated that their operations are almost identical to that of standard vectors. This assumption is found to be true for component-wise operations such as addition, equality, scalar multiplication and the dot product. However, it does not hold true for whole vector operations such as vector multiplication and conjugation. We can define these operations as follows.

Consider two quaternions $\mathbf{q}_1 = x_1 i + y_1 j + z_1 k + w_1$ and $\mathbf{q}_2 = x_2 i + y_2 j + z_2 k + w_2$. We define the quaternion multiplication, or Hamiltonian product, $\mathbf{q}_1\mathbf{q}_2$ as

$$\begin{aligned} \mathbf{q}_1\mathbf{q}_2 = &(w_1x_2 + x_1w_2 + y_1z_2 - z_1y_2)i \\ &+ (w_1y_2 - x_1z_2 + y_1w_2 + z_1x_2)j \\ &+ (w_1z_2 + x_1y_2 - y_1x_2 + z_1w_2)k \\ &+ (w_1w_2 - x_1x_2 - y_1y_2 - z_1z_2). \end{aligned} \tag{2.1.10}$$

The conjugate of $\mathbf{q} = (\mathbf{v}, w)$, with $\mathbf{v} = (x, y, z)$, is defined as

$$\mathbf{q}^* = -xi - yj - zk + w = (-\mathbf{v}, w), \tag{2.1.11}$$

and the normalised version of $\mathbf{q}$, called a unit quaternion, is

$$\hat{\mathbf{q}} = \frac{\mathbf{q}}{\|\mathbf{q}\|}, \tag{2.1.12}$$

where $\|\mathbf{q}\| = \sqrt{\mathbf{q}\mathbf{q}^*} = \sqrt{x^2 + y^2 + z^2 + w^2}$ is the norm of the quaternion.

We can also express any 3-vector as a purely imaginary quaternion by setting the imaginary vector $\mathbf{v}$ to the vector we wish to operate on and keeping the scalar part as zero, such that

$$\mathbf{q}(\mathbf{x}) = (\mathbf{x}, 0) = xi + yj + zk + 0. \tag{2.1.13}$$

Using this representation we can perform vector rotations using standard quaternion multiplication. In order to do this it is required that the quaternions representing

rotation are normalised. A rotation of angle θ about the axis $\mathbf{v}$ can be expressed as a quaternion of the form

$$\mathbf{q} = \left(\mathbf{v}\sin(\tfrac{\theta}{2}),\quad \cos(\tfrac{\theta}{2})\right). \tag{2.1.14}$$

Let us consider the case of rotating vector $\mathbf{x}_B$ from body to world coordinates, where the rotation is described by the unit quaternion $\hat{\mathbf{q}}$. We can express this transformation as

$$\mathbf{x}_W = \hat{\mathbf{q}}\mathbf{q}(\mathbf{x}_B)\hat{\mathbf{q}}^*, \tag{2.1.15}$$

where the vector $\mathbf{q}(\mathbf{x}_B)$ is a vector expressed in quaternion form as in equation 2.1.13. The output vector $\mathbf{x}_W$ is found to be the vector part of the quaternion, ignoring any value in the scalar component. We can also express this rotation in matrix form:

$$\begin{aligned}\mathbf{x}_W &= \mathbf{M}(\mathbf{q})\mathbf{x}_B \\ &= \frac{1}{\|\mathbf{q}\|}\begin{bmatrix} 1-2y^2-2z^2 & 2(xy-wz) & 2(xz+wy) \\ 2(xy+wz) & 1-2x^2-2z^2 & 2(yz-wx) \\ 2(xz-wy) & 2(yz+wx) & 1-2x^2-2y^2 \end{bmatrix}\mathbf{x}_B,\end{aligned} \tag{2.1.16}$$

where the matrix $\mathbf{M}(\mathbf{q})$ is the matrix representation of the rotation, and is directly related to the rotation matrix in equation 2.1.6.

We can convert between the quaternion and rotation matrix representations of a rotation. Converting from a quaternion to the equivalent rotation matrix is performed using the $\mathbf{M}(\mathbf{q})$ matrix defined in equation 2.1.16 and the scaling factor $1/||\mathbf{q}||$. In order to convert a rotation matrix to the equivalent quaternion representation we consider a pure rotation matrix $\mathbf{R}$ with a determinant of 1 and components r_{ij}. The conversion can then be expressed as

$$\hat{\mathbf{q}} = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} \frac{r_{32}-r_{23}}{4w} \\ \frac{r_{13}-r_{31}}{4w} \\ \frac{r_{21}-r_{12}}{4w} \\ \frac{\sqrt{1+r_{11}+r_{22}+r_{33}}}{2} \end{pmatrix}. \tag{2.1.17}$$

Quaternions provide a compact method of representing the attitude of a rigid body. Furthermore, they do not suffer from singularities and provide a slightly more intuitive way of thinking about rotation compared to the DCM. Quaternions exhibit another useful feature which is that if we use them in an iterative optimisation framework, such as bundle adjustment, we do not need to include a special parametrisation in order to impose the unity constraint. We can rather simply renormalise the quaternion after each iteration of the optimisation algorithm [11]. This fact makes quaternion optimisation less computationally intensive than optimising for a DCM where we have to ensure matrix orthogonality.

For these reasons we decide to use quaternions throughout our framework. Even in part where we refer to rotation matrices they are implemented as quaternions and only shown in matrix form to ease the description or understanding of the mathematics. For a more complete derivation and a full list of quaternion operations the user is referred to the literature [11, 53].

2.2 Inertial Sensors on Mobile Devices

Since the main purpose of inertial measurement on mobile devices is for gesture recognition or simple games, the sensors are not a good fit for odometry applications where greater accuracy is required. Some of the biggest shortcomings of the sensors on mobile devices are their low sample rate and the need for the sensors to be low cost. Most commercial inertial measurement units (IMUs) can sample in excess of 1kHz while the sensors on mobile devices, such as the iPhone, generally have a maximum sample rate of 120Hz. In our system we can only use 60Hz before the EKF begins to take up a significant amount of processing time. The inertial sensors found in mobile devices also suffer from bias, poor calibration and a low signal-to-noise ratio.

The following simple error model [45] can be used to describe the measurements obtained from 3-axis sensors such as the ones in the iPhone:

$$\mathbf{y}_k = s(\mathbf{I}_3 + \mathbf{M}_{\text{sa}})\mathbf{u}_k + \mathbf{b} + \mathbf{n}, \tag{2.2.1}$$

where $\mathbf{y}_k$ and $\mathbf{u}_k$ are the sensor output and input values respectively. The input vector is affected by a scale factor s as well as a misalignment and individual skew factor, defined by $\mathbf{M}_{\text{sa}}$. It is also affected by an additive bias $\mathbf{b}$ and additive Gaussian white noise $\mathbf{n}$. The misalignment $\mathbf{M}_{\text{sa}}$ and bias $\mathbf{b}$ must be found with the use of a calibration procedure.

2.2.1 Noise and Calibration

The noise of all the inertial sensors is assumed to be zero mean additive Gaussian noise. This noise is independent of any motion and thus cannot be calibrated out. In order to determine the noise characteristics for the inertial sensors of the iPhone 5, we can leave the device on a stationary surface for a few minutes and record the inertial data. As there is no actual movement it is clear that deviations in the sensor readings during this period are induced due to sensor and environmental noise.

The inertial sensors on the iPhone 5 have been pre-calibrated by the manufacturers in the factory. This means that the bias $\mathbf{b}$ and misalignment matrix $\mathbf{M}_{\text{sa}}$ should be relatively small, as is confirmed by our experiments.

Figures 2.4 and 2.5 show histograms of the accelerometer and gyroscope sensor readings with corresponding Gaussian approximations of the distribution. It is important that the noise can be characterised by a Gaussian distribution as the EKF requires the use of a Gaussian noise model. The non-zero mean of the z-axis accelerometer readings is due to the effect of gravity.

By analysing these Gaussian distributions, under the assumption that the x, y and z components are uncorrelated, we can determine the covariance matrices for

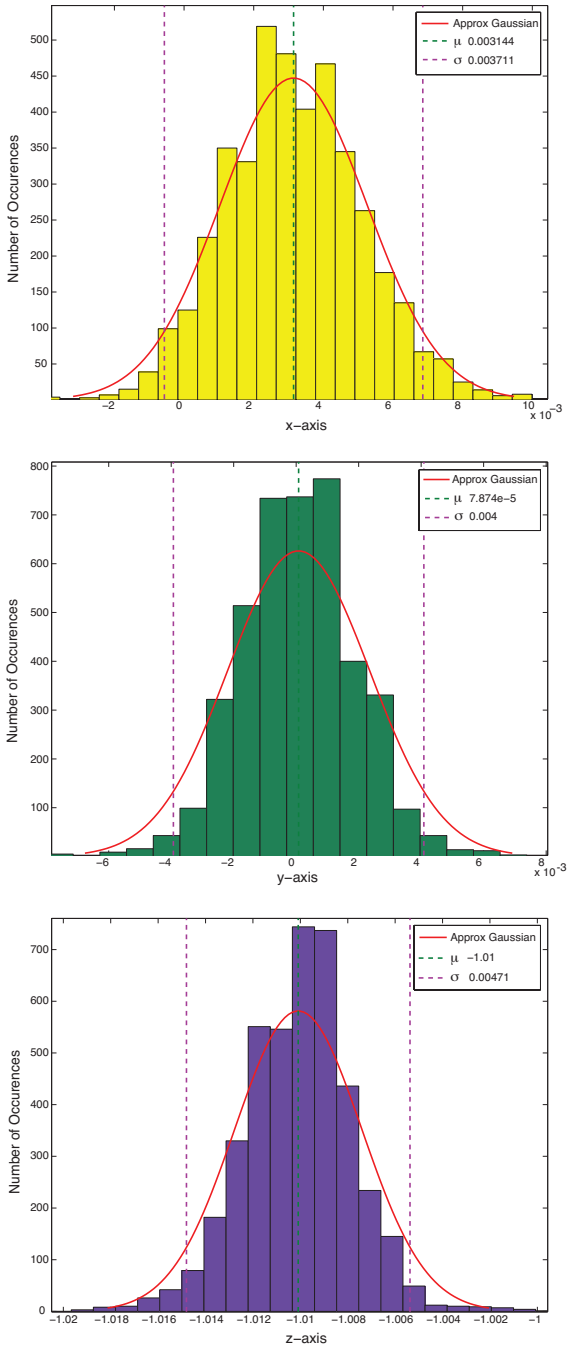


Figure 2.4: Measured and approximated Gaussian noise distribution of the MEMS accelerometer on the iPhone 5. The unit of measure of these sensor readings (on the horizontal axis) is standard gravity g.

the accelerometer (A_α) and gyroscope (A_ω) readings:

$$A_\alpha = \begin{bmatrix} 1.38 \times 10^{-5} & 0 & 0 \\ 0 & 1.60 \times 10^{-5} & 0 \\ 0 & 0 & 2.22 \times 10^{-5} \end{bmatrix}, \tag{2.2.2}$$

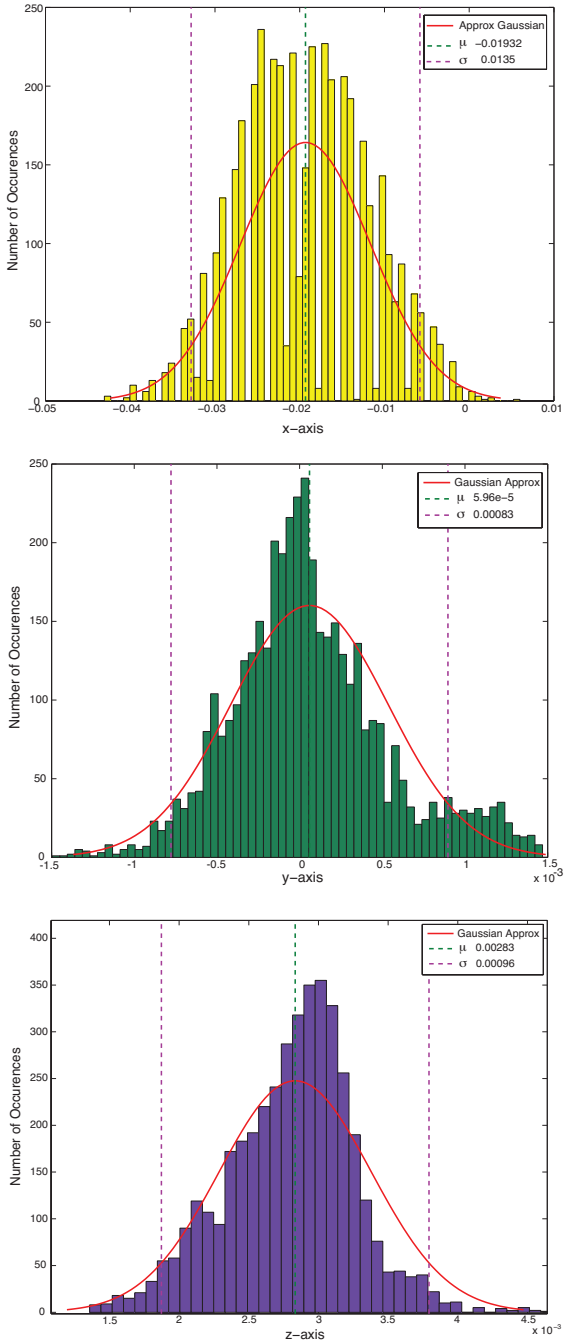


Figure 2.5: Measured and approximated Gaussian noise distribution of the gyroscope on the iPhone 5. These sensor readings (on the horizontal axis) are in rad $\cdot\, s^{-1}$.

$$A_\omega = \begin{bmatrix} 1.82 \times 10^{-4} & 0 & 0 \\ 0 & 6.89 \times 10^{-7} & 0 \\ 0 & 0 & 9.22 \times 10^{-7} \end{bmatrix}. \tag{2.2.3}$$

The magnetometer readings have a non-zero mean which is due to the magnetic field strength induced by the Earth and surrounding ferromagnetic objects. We do not show graphs of the magnetometer as it does not exhibit Gaussian noise, but is rather affected by disturbances in the magnetic field as we discuss in section 2.2.4.

2.2.2 Accelerometer

A linear accelerometer is a device which measures acceleration forces acting on it. MEMS accelerometers, such as the one found in the iPhone 5, use Newton's second law of motion ($F = ma$) to determine acceleration from the forces acting on a proof mass.

Accelerometers are useful for pose estimation and can be used to extract the device's pitch and roll relative to gravity. Accelerometers do not experience drift and thus provide a reliable way to estimate two DoF of the attitude. However, when influenced by fast or large motion the gravity vector can become tainted and thus the pose estimate can become unstable.

The iPhone 5 uses an STMicroelectronics LIS331DLH linear accelerometer, which measures acceleration relative to gravity. Thus we need to convert the acceleration to metric units by multiplying each component by $9.81ms^{-2}$.

2.2.3 Gyroscope

Gyroscopes measure the angular velocity in radians per second by exploiting the principles of conservation of angular momentum. Unlike accelerometers, gyroscopes handle fast motion well and can be integrated over short time periods to provide a 3DoF rotation estimate. That said, however, MEMS gyroscopes, such as the STMicroelectronics L3G4200D found in the iPhone 5, are less accurate than mechanical or optical gyroscopes.

MEMS gyroscopes often suffer from bias drift and calibration errors. As seen in figure 2.5 the gyroscope has a negligible bias term (likely due to compensation by the manufacturers). The bias drift of the gyroscope is something which cannot be corrected for by considering the gyroscope readings alone, and this is where a process called sensor fusion helps. By coupling measurements from the gyroscope with those from the accelerometer we can significantly reduce the effects of drift on the gyroscope measurements. This approach is described in section 2.3.

2.2.4 Magnetometer

Magnetometers measure the strength and direction of any magnetic field, and are usually used to measure the Earth's magnetic field.

Apart from the typical sensor errors such as bias and noise, magnetometers also experience distortions due to ferromagnetic objects near the sensor. These distortions are classed as either hard iron or soft iron distortions. Hard iron distortions

are caused by objects which exhibit a constant additive magnetic field and soft iron distortions are caused by objects which distort the magnetic field but do not generate one [21]. The magnetometer in the iPhone is pre-calibrated against hard iron distortions created by the components of the device, however it needs to be calibrated before each use as well. The reason for this is that the Earth's magnetic field is not uniform. The magnetometer sensor reading is dependent on altitude and location as well as local magnetic distortions in the immediate environment.

A description of the full magnetometer calibration procedure is beyond the scope of this thesis. Here we will describe the calibration of two of the three axes. This process can be extended to full 3D calibration by using a sphere and ellipsoids. Consider the case of a magnetometer with only two DoF in the x- and y-axes. If we subject a calibrated magnetometer to a full rotation in the xy-plane and plot the samples we should see a perfect circle centred at the origin of the Cartesian plane. However, if the magnetometer is not calibrated this circle will be skewed into an ellipse and offset from the origin, as in figure 2.6.

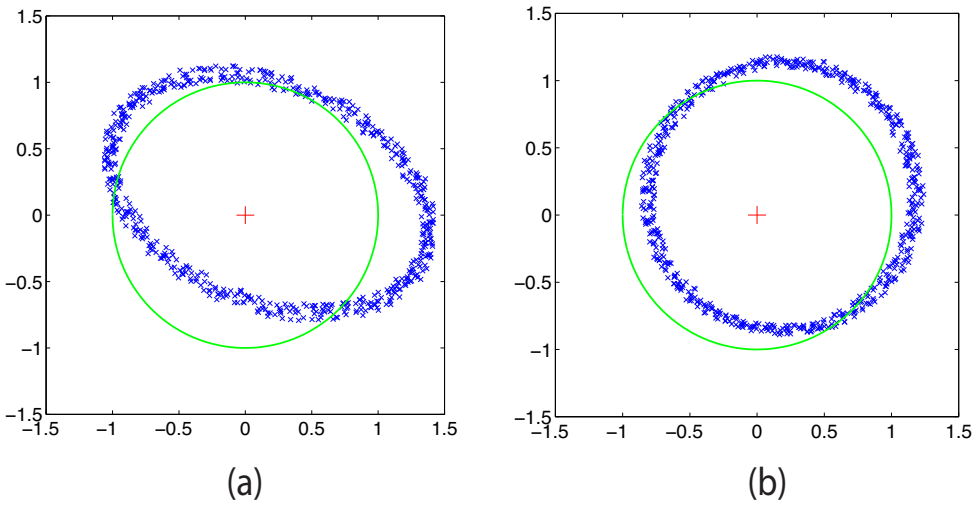


Figure 2.6: The effect of magnetic distortions on the magnetometer readings in our 2D motion example: (a) the skewing effect of soft iron distortions and the offset from hard iron distortions; (b) the same magnetometer with soft iron distortions calibrated out, and the offset caused by hard iron still visible. In both plots the red cross shows the centre point and the green circle shows the reference which calibration tries to achieve.

The offset is caused by hard iron distortion and the skew is caused by soft iron distortions. Our calibration procedure calculates the rotation required to align the major and minor axes to the x- and y-axes, as well as the scaling factors and offset values to ensure the final result is a circle centred at the origin, as depicted in figure 2.7.

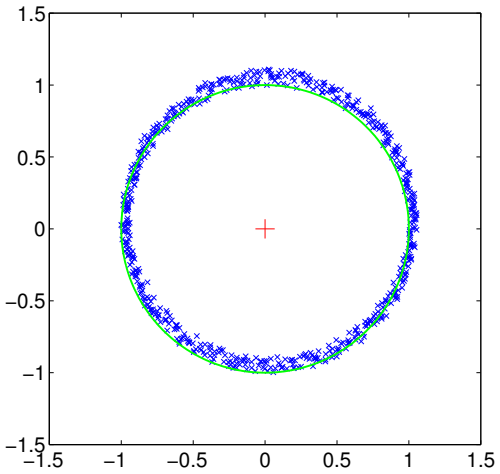


Figure 2.7: Final calibrated readings from the magnetometer when affected by the 2D motion we described.

2.2.4.1 iPhone Calibration Errors

We performed this simple 2D calibration on the iPhone 5 to ensure the calibration was correct, but we discovered that even after calibration there were problems with the magnetometer data being returned.

In order to verify the on-board calibration algorithms we subjected the sensor to a set of 2D rotations about the z-axis. The calibrated sensor readings obtained from this procedure did not produce the expected outcome, as described above. This brought to our attention that a fault exists within the internal calibration routine found in iOS 7.0. The sample points, shown in figure 2.8, do not form a perfect circle, but rather appear to create discontinuities from time to time. This is confirmed by observing the magnetometer readings for a single axis as we subject the sensor to various soft iron distortions. The CoreMotion library does not provide a way to directly access the raw magnetometer sensor data and thus it is not possible to perform a full calibration ourselves.

As we demonstrate later, discontinuities appear in the sensor output at the instances where soft iron distortion affects the field. These distortions should appear as gradual changes in the magnetometer readings, as the metal object is moved closer to the magnetometer, rather than discontinuities. Fortunately, this discontinuous behaviour allows us to detect the exact point the magnetic field is affected by a distortion and thus we can compensate for it without the need to determine skew and scaling factors.

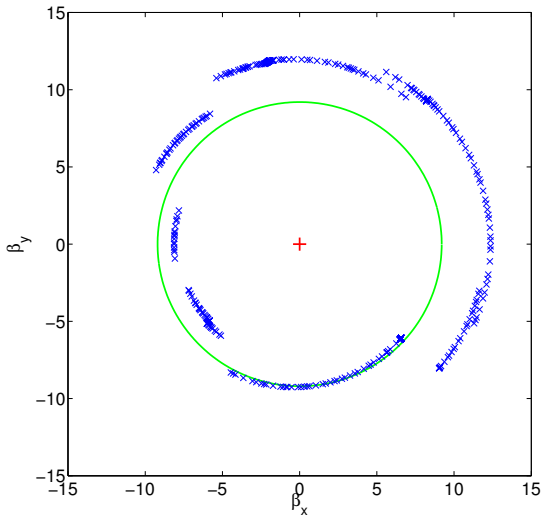


Figure 2.8: The magnetometer readings after calibration on the iPhone 5. It can clearly be seen that the field experiences discontinuities instead of producing the expected circle around the origin.

2.2.4.2 Magnetometer Correction

In order to use the magnetometer we need to ensure that the sensor values exhibit the properties of a properly calibrated magnetometer, as discussed at the start of section 2.2.4. We exploit the discontinuities in the magnetometer readings in order to correct for the erroneous calibration.

In order to detect discontinuities and correct for the offset they produce in the magnetometer data, we consider a 2D rotation of the device in the xy-plane. Under this motion, and assuming no disturbances, the magnitude of the magnetic field will remain constant. As the magnitude remains constant, and there is no motion along the z-axis, it is obvious that the x- and y-components of the magnetic field strength will change in such a way as to keep the magnitude constant. For the motion we are considering, this change can be described by the sine and cosine of the angle of the rotation.

From this observation it is clear that the magnetic field strength is continuous. Adding the assumption of Gaussian noise and small motion we can further define the change in the magnetic field as a Gaussian random walk, as exemplified in figure 2.9. It follows that the next sensor reading β_{k+1} for any axis can be described by the current reading and a Gaussian distribution, such that $\beta_{k+1} \sim \mathcal{N}(\beta_k, \sigma^2)$ where σ is some standard deviation describing the noise. We can further extend this model to handle larger motions by predicting the next location using a decaying motion model, such that

$$\tilde{\beta}_{k+1} = \beta_k + \triangle\beta^k_{k-1}(1 - \exp(\frac{-\tau}{t})), \tag{2.2.4}$$

where $\Delta\beta^k_{k-1}$ is the previous rate of change of magnetic field strength, β_k is the

magnetic field measured by the magnetometer in the body frame of reference and τ is a tuning parameter describing the motion. We found that $\tau = 0.6$ works for most motions. Using equation 2.2.4 we can now describe the random walk process by $\beta_{k+1} \sim \mathcal{N}(\tilde{\beta}_{k+1}, \sigma^2)$.

If we now consider the case where disturbances occur during the actual motion, we can see that it becomes possible to determine if the motion is affected by a disturbance or if it is within the bounds assumed by our random walk model. We can determine this by examining the probability that the measurement is generated by the distribution $\mathcal{N}(\tilde{\beta}_{k+1}, \sigma^2)$. If the probability is less than approximately 0.68, or outside of one standard deviation, we may deduce that there is a disturbance. This principle is depicted in figure 2.9.

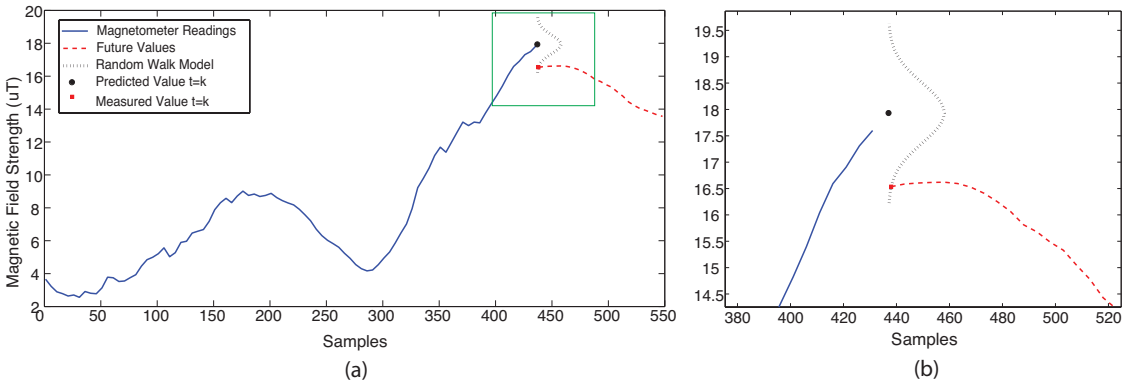


Figure 2.9: Our Gaussian random walk model with respect to measurements from a single magnetometer axis. The prediction is indicated by a black circle and the actual measurement by a red square. The red dashed line indicates the future measurements which are affected by a disturbance bias. Figure (b) represents a close up of the green box in figure (a), showing the random walk model for the current time step.

This correction model is only possible when magnetic field disturbances appear as discontinuities, which is the case with the iPhone magnetometer due to its faulty factory calibration. Once it is determined that the motion is due to a disturbance we can subtract the bias added by this disturbance from the measured signal. The result is a corrected magnetometer reading which is not affected by magnetic disturbance. In order to test our filter, we left the device in a stationary position away from metallic objects and then subjected it to magnetic disturbances. The results of this test and the success of the filter can be seen in figure 2.10.

We can further increase the accuracy of our approach by scaling the standard deviation σ according to the motion. If the motion is fast, then there is a larger likelihood that the device is accelerating and that the velocity will not allow us to predict the new location accurately. In this case we can increase the standard deviation to near a uniform distribution when the motion is fast. If the previous motions are very small we can assume that future motions will also remain small

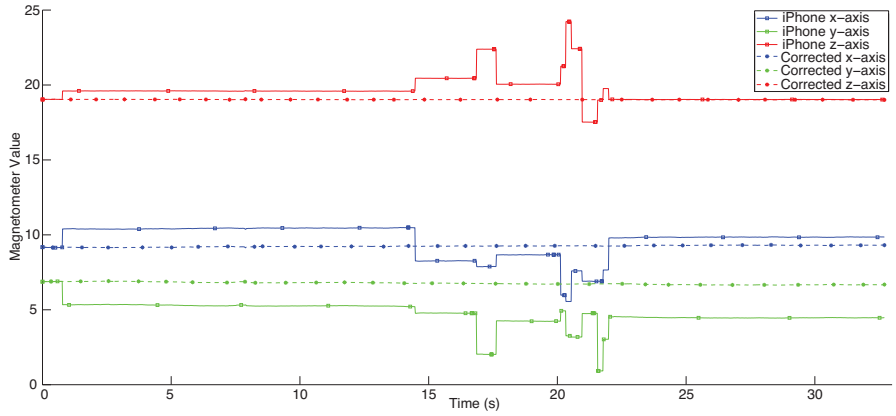


Figure 2.10: The original and corrected magnetometer readings for a stationary device. The unwanted discontinuities are caused by moving a metal object around the location of the phone.

or will gradually begin increasing. By extending the model in this way we can then detect small disturbances during small motion while still being able to handle larger motions. Due to the nature of our application it is unlikely that the device will experience large motions, as these will negatively affect the visual processing stages discussed in chapter 4. We update the standard deviation according to

$$\sigma = 0.05 + |\Delta m_{k-1}| \left(1 - \exp(\frac{-1}{\tau_m})\right), \tag{2.2.5}$$

where Δm_{k-1} is the rate of change of the magnetometer readings at the previous time step and τ_m is a constant describing how fast we expect motion to decay. In our application we set $\tau_m = 1.2$, which we found to work sufficiently well.

We can confirm that our filter is working by analysing the magnitude of the magnetometer readings as the phone is moved around in 3-space. We can see from figure 2.11 that the corrected magnitude is constant, as we expect for motion over a small area. This shows that the disturbances caused by metallic objects in the area of the magnetometer are now corrected and have a smaller influence on the magnetometer readings.

The results of a more complete test of our filter's accuracy and robustness is presented in chapter 6.

2.3 Proposed Sensor Fusion Approach

In order to obtain a robust and accurate pose estimate from the inertial sensors we employ a technique known as sensor fusion. Sensor fusion refers to the process of combining multiple different measurements together in order to exploit the strengths of the various sensors and mitigate their faults.

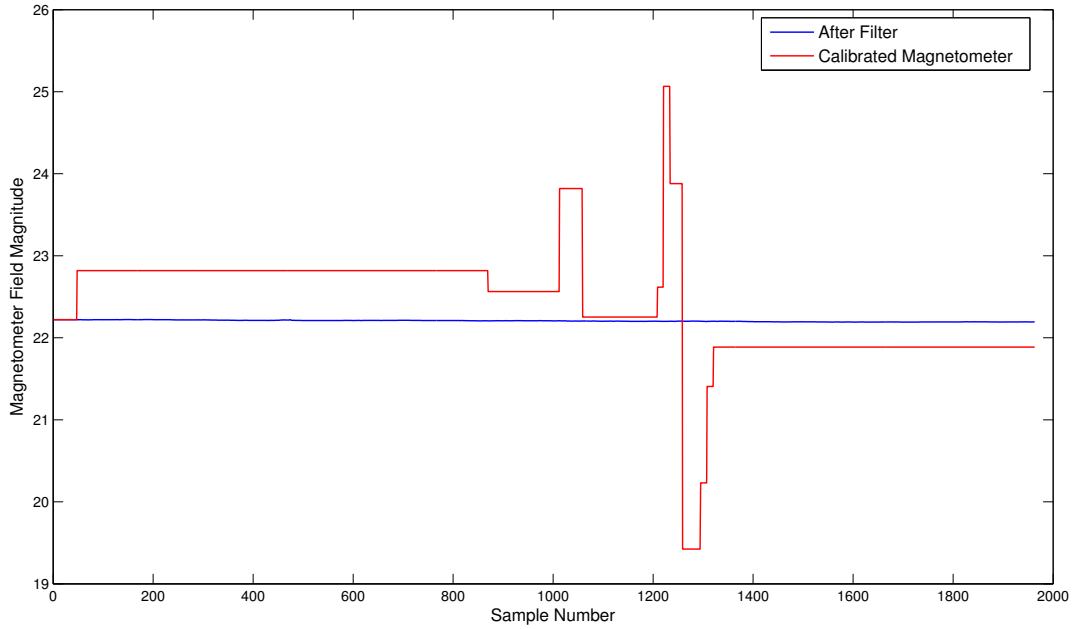


Figure 2.11: Magnetic field magnitude, for large motion over a small area, before and after applying our filter.

In the case of the sensors on a mobile device we require sensor fusion to reduce the effect of drift on the gyroscope, and to decrease the noise on the accelerometer and magnetometer. Furthermore, sensor fusion allows us accurately estimate the pose even under fast motion where the magnetometer and accelerometers become less accurate.

In this section we derive our Kalman filter and motivate our choices for the layout of the filter.

2.3.1 The Extended Kalman Filter

The Kalman filter is a special type of recursive Bayes estimation where the states are assumed to be related linearly and the noise normally distributed. It estimates the system state with respect to time using noisy measurements from the actual system and a mathematical model of the system dynamics. The recursive algorithm used in a Kalman filter consists of two main steps: prediction and correction. The predication step uses a mathematical model of the process and the previous estimate to predict what the next value of the states will be. The correction step updates this estimate and the uncertainty using the actual measurements [55].

As the attitude process model is nonlinear, we cannot make use of the standard Kalman filter that assumes a linear process. For this reason we make use of an extended Kalman filter. The EKF extends the Kalman filter by using a first order Taylor expansion of the process and measurement model in order to allow a linear

approximation of the nonlinear function to be used. For this approximation it is important for the process and measurement models to be differentiable.

The EKF prediction step can be described by the following equations:

$$\mathbf{x}_k^- = \mathbf{f}_{k-1}(\mathbf{x}_{k-1}^+, \mathbf{u}_{k-1}, \mathbf{0}), \tag{2.3.1}$$

$$\mathbf{P}_k^- = \mathbf{F}_{k-1}\mathbf{P}_{k-1}^+\mathbf{F}_{k-1}^T + \mathbf{Q}_{k-1}, \tag{2.3.2}$$

where $\mathbf{x}$ is the state vector and $\mathbf{F}_{k-1}$ is the Jacobian of the state transition function $\mathbf{f}$ evaluated at the current system state. The matrix $\mathbf{F}$ is known as the state transition matrix. The matrix $\mathbf{P}$ is the estimated state covariance matrix and $\mathbf{Q}$ is the process noise covariance matrix. The superscript minus represents the a priori estimate and the superscript plus represents the a posteriori estimate (from the previous time step in the equations above).

The correction (update) step of the EKF can be described by the following equations:

$$\mathbf{K}_k = \mathbf{P}_k^-\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}_k^-\mathbf{H}_k^T + \mathbf{R}_k)^{-1}, \tag{2.3.3}$$

$$\mathbf{y}_k = \mathbf{z}_k - \mathbf{h}(\mathbf{x}_k^-), \tag{2.3.4}$$

$$\mathbf{x}_k^+ = \mathbf{x}_k^- + \mathbf{K}_k\mathbf{y}_k, \tag{2.3.5}$$

$$\mathbf{P}_k^+ = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\,\mathbf{P}_k^-, \tag{2.3.6}$$

where $\mathbf{H}_k$ is known as the observation matrix, and is the Jacobian of the measurement function $\mathbf{h}$ evaluated at the previous system state. $\mathbf{z}_k$ is a noisy measurement of the system state, and $\mathbf{R}$ is the measurement covariance matrix.

2.3.2 Kalman Filter Design

Following on from the Kalman filters derived by Von Marcard [54] and Marins et al. [30] we use a quaternion to represent the attitude of the device, and a 3-vector to represent the angular body velocities of the device. As was shown by Von Marcard [54], this leads to a simplified mathematical model and a linear output model, however it does require that some pre-processing is done on the magnetometer and accelerometer measurements.

Our EKF implementation consists of two stages: a quaternion measurement stage and the actual EKF stage, as can be seen in figure 2.12. The quaternion measurement stage (TRIAD) takes in the accelerometer $\mathbf{a}_k$ and magnetometer $\mathbf{m}_k$ readings and synthesises a quaternion measurement $\mathbf{q}_{\text{triad}}$, as we discuss in section 2.3.2.4. The Kalman filter stage takes in measurements of the attitude and angular velocity $\boldsymbol{\omega}_k$ and estimates the current device attitude $\mathbf{q}_{\text{ekf}}$ and angular velocity $\boldsymbol{\omega}_{\text{ekf}}$. Overall the layout shown in figure 2.12 leads to reduced computational complexity with little effect on the accuracy of the results. In the remainder of this section we derive the system model for our Kalman filter, and the details required for implementation.

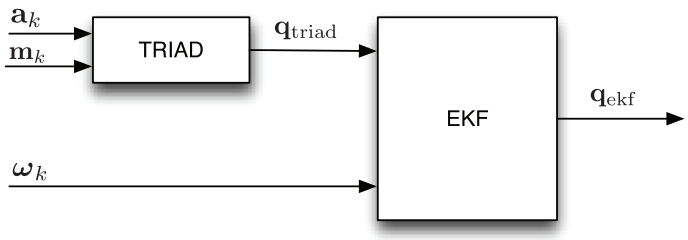


Figure 2.12: Our extended Kalman filter layout.

2.3.2.1 The State Vector

Before we can describe the system mathematically we need to define a state vector. The state vector represents the actual attitude as well as the internal dynamics of the system.

Thus we define the state vector of our process model as

$$\mathbf{x} = \begin{pmatrix} p & q & r & x & y & z & w \end{pmatrix}^T, \tag{2.3.7}$$

where p, q and r represent the angular body velocities about the x-, y- and z-axes respectively, and x, y, z and w represent the quaternion components of the device's attitude.

Our state vector is the bare minimum needed to model the orientation and dynamics of the system. It is common practice to append error and bias terms to the state vector in order to improve accuracy and better compensate for bias. However, the additional computational complexity added by these states outweighs the benefit of estimating them, and thus we do not use that approach.

2.3.2.2 System Equations

The prediction stage of the Kalman filter relies on having a realistic mathematical model of the system in order to predict the next state. In order to derive this state space representation of the system we need to develop a model of the system dynamics. The system equations describe the evolution of the state vector with respect to time and can thus be represented as the first derivative of the state vector $\mathbf{x}$.

As the motion we are trying to estimate is unpredictable, and only bounded by the constraints of human motion, we employ the dynamic motion model used by Von Marcard [54] which describes the angular accelerations as a random walk process. Using this model we can express the state equations for the angular acceleration as

$$\begin{aligned} \dot{p} &= -\frac{1}{\tau}p + \frac{1}{\tau}s_p, \\ \dot{q} &= -\frac{1}{\tau}q + \frac{1}{\tau}s_q, \\ \dot{r} &= -\frac{1}{\tau}r + \frac{1}{\tau}s_r, \end{aligned} \tag{2.3.8}$$

where τ is a time constant describing the rate of change of the angular motion, and s_p, s_q, s_r represent the driving noise of the process which is shown by Von Marcard to be independent and white. For simplicity we implement only a single time constant τ (in fact, it is found that the best performance is reached when every axis of rotation has the same model parameters).

Next we find the system equations which describe the change in the quaternion states with respect to the current orientation and angular rates. We can relate the quaternion rate to the angular body rate through a simple quaternion product, such that

$$\dot{\mathbf{q}} = \frac{1}{2}\boldsymbol{\omega}\hat{\mathbf{q}}, \tag{2.3.9}$$

where $\boldsymbol{\omega}$ is the angular rate vector expressed as a quaternion, and $\hat{\mathbf{q}}$ is a unit quaternion. Using equation 2.3.9 we can express the quaternion system equations as

$$\begin{aligned}
\dot{x} &= \frac{1}{2}(wp - zq + yr), \\
\dot{y} &= \frac{1}{2}(zp + wq - xr), \\
\dot{z} &= \frac{1}{2}(-yp + xq + wr), \\
\dot{w} &= \frac{1}{2}(-xp - yq - zr).
\end{aligned} \tag{2.3.10}$$

2.3.2.3 Pose Prediction

Assuming the EKF has been initialised, we can propagate the current state vector forward in time using equations 2.3.8 and 2.3.10. In order to be able to implement the EKF we use a zero order hold and Euler's method.

The zero order hold fixes the value of a function between two sampling points, such that

$$f(t) = f(t_k),\ t_k \leq t < t_{k+1}. \tag{2.3.11}$$

By utilising the zero order hold we can make use of Euler's method for approximating derivatives in a discretised system,

$$\dot{x} = \frac{x_{k+1} - x_k}{\delta t}, \tag{2.3.12}$$

where δt is the sampling period in seconds. The sample rate in our system is 60Hz, thus $\delta t = 0.01667$s.

By expressing the derivatives in the system equations using Euler's method and reordering, we obtain the following discretised process equations:

$$\begin{aligned}
p_{k+1} &= (1 - \frac{\delta t}{\tau})p_k + \frac{\delta t}{\tau}s_{pk}, \\
q_{k+1} &= (1 - \frac{\delta t}{\tau})q_k + \frac{\delta t}{\tau}s_{qk}, \\
r_{k+1} &= (1 - \frac{\delta t}{\tau})r_k + \frac{\delta t}{\tau}s_{rk},
\end{aligned} \tag{2.3.13}$$

$$
\begin{aligned}
x_{k+1} &= \frac{\delta t}{2}(w_k p_k - z_k q_k + y_k r_k) + x_k, \\
y_{k+1} &= \frac{\delta t}{2}(z_k p_k + w_k q_k - x_k r_k) + y_k, \\
z_{k+1} &= \frac{\delta t}{2}(-y_k p_k + x_k q_k + w_k r_k) + z_k, \\
w_{k+1} &= \frac{\delta t}{2}(-x_k p_k - y_k q_k - z_k r_k) + w_k.
\end{aligned} \tag{2.3.14}
$$

The process model defined above is represented as the function $\mathbf{f}$ in the EKF definition stated in equation 2.3.1. Using this process model and the previous state estimate we can estimate the a priori state $\mathbf{x}_k^-$ as in equation 2.3.1. In order to determine the covariance of our estimate we first need to linearise the process model by taking partial derivatives of the state equations and evaluating them at the current estimate, to obtain

$$
\mathbf{F}_k = \left.\frac{\delta \mathbf{f}_k}{\delta \mathbf{x}}\right|_{\mathbf{x}_k} = \begin{bmatrix}
1 - \frac{\delta t}{\tau} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 - \frac{\delta t}{\tau} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 - \frac{\delta t}{\tau} & 0 & 0 & 0 & 0 \\
\frac{\delta t}{2} w_k & -\frac{\delta t}{2} z_k & \frac{\delta t}{2} y_k & 1 & \frac{\delta t}{2} r_k & -\frac{\delta t}{2} q_k & \frac{\delta t}{2} p_k \\
\frac{\delta t}{2} z_k & \frac{\delta t}{2} w_k & -\frac{\delta t}{2} x_k & -\frac{\delta t}{2} r_k & 1 & \frac{\delta t}{2} p_k & \frac{\delta t}{2} q_k \\
-\frac{\delta t}{2} y_k & \frac{\delta t}{2} x_k & \frac{\delta t}{2} w_k & \frac{\delta t}{2} q_k & -\frac{\delta t}{2} p_k & 1 & \frac{\delta t}{2} r_k \\
-\frac{\delta t}{2} x_k & -\frac{\delta t}{2} y_k & -\frac{\delta t}{2} z_k & -\frac{\delta t}{2} p_k & -\frac{\delta t}{2} q_k & -\frac{\delta t}{2} r_k & 1
\end{bmatrix}. \tag{2.3.15}
$$

Using equations 2.3.2 and 2.3.15, the process noise covariance $\mathbf{Q}$ and the current state estimate we can now update the a priori covariance estimate.

2.3.2.4 Measurement and Correction

Once we have predicted what we expect the next system state to be, as well as the uncertainty of this prediction, we need to correct for this estimate using the actual measured values $\mathbf{z}$. Using these measurement values and equations 2.3.3 to 2.3.6 we can calculate the corrected system state estimate.

We define the measurement vector $\mathbf{z}_k$ to have the same elements as the state vector. By doing this we ensure that the output equations of the system are linear, and thus reduce the computational complexity of the filter. More specifically if the measurement vector has the same elements as the state vector, we see from equation 2.3.4 that the output equation is just the identity and thus the Jacobian $\mathbf{H}_k$ is the identity matrix. This approach is followed by Von Marcard [54] as well as Marins et al. [30].

Obtaining a measurement for the first three states is easy as it is taken directly from the gyroscope of the mobile device. However, it is not possible to directly obtain a measurement of the orientation as a quaternion. We can, however, synthesise an orientation measurement from the accelerometer and magnetometer measurements. For this we use the TRIAD algorithm, as it was shown to have similar performance to more expensive, iterative optimisation based approaches such as QUEST when only two vector measurements are used [54]. Furthermore, the TRIAD algorithm requires only one very accurate vector measurement as the second measurement is only used, with the cross product, to generate an orthogonal basis vector for the coordinate frame. This fits well with the fact that the magnetometer measurement is more susceptible to noise than the accelerometer.

The TRIAD algorithm makes use of two vector measurements and two reference vectors. The algorithm uses one vector measurement as a basis vector for the frame, and then uses vector orthogonality principles to create the orthogonal frame. Doing this for both the measurements and the reference vectors we obtain two orthogonal frames, which we can compare in order to find the relative rotation between them.

We use the acceleration as the first basis vector for our body frame such that

$$\mathbf{r}_1 = \frac{\mathbf{a}_B}{\|\mathbf{a}_B\|}. \tag{2.3.16}$$

The second basis vector is then constructed using $\mathbf{r}_1$ and the magnetometer measurement, such that $\mathbf{r}_2$ is orthogonal to $\mathbf{r}_1$ and orientated correctly:

$$\mathbf{r}_2 = \frac{\mathbf{r}_1 \times \mathbf{m}_B}{\|\mathbf{r}_1 \times \mathbf{m}_B\|}. \tag{2.3.17}$$

In order to complete the body triad we create the third basis vector to be orthogonal to both $\mathbf{r}_1$ and $\mathbf{r}_2$:

$$\mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2. \tag{2.3.18}$$

Using the same approach we construct a world triad. However, instead of using absolute world measurements such as gravity and the Earth's magnetic field we use the measurements we obtain during system initialisation. These are equivalent to the body vectors at the first frame. Thus our world frame of reference is given by

$$\mathbf{s}_1 = \frac{\mathbf{a}_W}{\|\mathbf{a}_W\|}, \tag{2.3.19}$$

$$\mathbf{s}_2 = \frac{\mathbf{s}_1 \times \mathbf{m}_W}{\|\mathbf{s}_1 \times \mathbf{m}_W\|}, \tag{2.3.20}$$

$$\mathbf{s}_3 = \mathbf{s}_1 \times \mathbf{s}_2. \tag{2.3.21}$$

Once we obtain the body and world frame triads we can calculate the orientation transformation $\mathbf{A}$ which maps the body frame to the world frame:

$$\begin{bmatrix}\mathbf{s}_1 & \mathbf{s}_2 & \mathbf{s}_3\end{bmatrix} = \mathbf{A}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3\end{bmatrix}. \tag{2.3.22}$$

As each triad represents a set of orthonormal basis vectors, we can represent their inverses by their transposes. Thus the orientation transformation $\mathbf{A}$ can be expressed as

$$\mathbf{A} = \begin{bmatrix} \mathbf{s}_1 & \mathbf{s}_2 & \mathbf{s}_3 \end{bmatrix} \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 \end{bmatrix}^T. \tag{2.3.23}$$

We can express the orientation measurement $\mathbf{A}$ as a quaternion using equation 2.1.16. Thus our measurement equation simply becomes

$$\mathbf{z}_k = \mathbf{x}_k + \mathbf{v}, \tag{2.3.24}$$

where $\mathbf{v}$ is the measurement noise vector with covariance matrix $\mathbf{R}$.

2.3.3 Initialisation and Noise Considerations

Unlike the Kalman filter, the extended Kalman filter is not an optimal estimator. This is due to the assumption that a first order approximation is sufficient to accurately model the system dynamics. So if the system model is incorrect, or not sufficiently parametrised by a first order approximation, then estimates from the filter may diverge. Furthermore, incorrectly initialising the state vector or covariance matrices may also cause the estimates to diverge.

It is also important to note that motion with a dynamic component over 30Hz will cause aliasing in our inertial measurements which in turn will reduce the accuracy of our EKF implementation as the high dynamics will not be captured by the sensors. This behaviour is based on Nyquist sampling theory, which states that the sampling rate needs to be at least double that of the highest frequency we wish to capture.

This being said the extended Kalman filter is still widely used and works well in many cases. Furthermore, the work done by Von Marcard [54], Marins et al. [30] and Aksoy [1] show successful implementations of the EKF in similar environments to ours.

In order to initialise the filter we assume the system to be at rest in the world coordinate system. Our initial state vector can thus be described as

$$\mathbf{x}_0 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}^T. \tag{2.3.25}$$

The estimation covariance matrix $\mathbf{P}$ is initialised as a diagonal matrix:

$$\mathbf{P} = \alpha \mathbf{I}_7, \tag{2.3.26}$$

where we choose $\alpha = 1$ and $\mathbf{I}_7$ represents the 7×7 identity matrix. If we initially set α to be too small it could lead to stability errors.

In order to set the process and measurement covariance matrices $\mathbf{Q}$ and $\mathbf{R}$ we use the values we obtained for the sensor noise in section 2.2.1. We also manually

adjust the values based on the work of Von Marcard [54], such that

$$\mathbf{Q} = \begin{bmatrix} 10^{-3} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10^{-3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10^{-3} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10^{-5} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10^{-5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10^{-5} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10^{-5} \end{bmatrix}, \tag{2.3.27}$$

$$\mathbf{R} = \begin{bmatrix} 10^{-3} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10^{-3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10^{-3} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10^{-4} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10^{-4} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10^{-4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10^{-4} \end{bmatrix}. \tag{2.3.28}$$

This concludes the discussion on our extended Kalman filter. In this chapter a brief description of the inertial sensors and their noise properties was provided. Furthermore, a description of our sensor fusion approach was supplied along with an explanation of the initialisation procedure. In the proceeding chapters we utilise the output of our sensor fusion system in order to assist various vision sub-systems, such as feature tracking and visual pose estimation.

Chapter 3

Feature Correspondence

Before we can apply visual SfM methods to a sequence of images we need to know how the images relate to one another. A key step in finding the relationship between images relies on finding points, in the images, which correspond to the same 3D features. These corresponding points allow us to determine the relationship between the various camera poses, as we describe in chapter 4.

Two main approaches toward determining image feature correspondences exist, namely detection and matching, and feature tracking. Each of these approaches can be further broken down into different methods which try to optimise the trade-offs between robustness, accuracy and speed. For pose estimation on a mobile device it becomes even more critical that we find the best balance between these three constraints as we have reduced processing power and lower quality sensors to take into account.

In this chapter we discuss the two types of feature correspondence approaches mentioned above. We further discuss methods of each of these which might be appropriate for use in a mobile pose estimation pipeline, and then propose a hybrid approach which we argue can provide better performance on mobile devices than each of the individual methods. We refrain from describing full implementations of other algorithms and rather provide a higher level overview on how they work. For a more complete description of the algorithms the reader is referred to the literature [42, 12, 29, 49, 2].

3.1 Detection and Matching

Feature detection and matching algorithms have been a main focal point in the field of computer vision for the past 25 years. Over this time a large array of vastly different algorithms have been developed, each of which have strengths in different application areas. The most notable feature detection and description algorithm is Lowe's scale invariant feature transform (SIFT) which is still seen as the gold standard measure some 15 years after it was published [27].

From the wide variety of feature detection, description and matching algorithms we choose to investigate ones which have shown good performance in reduced computational environments.

3.1.1 Feature Detection

The main goal of feature detection is to find salient and distinct points in an image, with the idea being that the same points should be found in another image of the same scene despite changes in camera position, illumination or scale.

Due to the fact that our pose estimation pipeline relies on video data and key frames, the requirements for the feature detection algorithm are somewhat reduced. The structure of our framework allows for the following simplifying assumptions to be made:

- key frames will have overlapping features;
- inter-frame motion will be relatively small;
- feature scale will change gradually over time.

3.1.1.1 FAST

The features from accelerated segment test (FAST) detector is by far the most simple and least computationally complex feature detection algorithm available. It was originally proposed by Rosten and Drummond as a method for identifying interest points in real-time applications with limited computational resources [42].

The FAST algorithm works by performing a test for a feature at pixel p by examining the 16 pixels on a Bresenham circle of radius 3 around p. The point p is determined to be a feature if the intensities of at least k contiguous pixels are all greater than the intensity of p by a pre-specified threshold t.

In the original FAST algorithm the value of k was chosen to be 12, however it has more recently been shown that using $k = 9$ provides better repeatability [43]. Values of t can vary depending on the application, but values between 40 and 60 are common choices.

In order to further speed up feature point detection, a simple discrimination test can be performed in order to quickly discard non-corner points. This test checks the values of pixels 1, 5, 9 and 13, as indicated in figure 3.1, as a feature can exist only if the intensity of at least three of these test points are different from the intensity of p by the threshold t [42].

The biggest disadvantage of the FAST algorithm comes from the fact that multiple interest points may be detected next to one another. This becomes problematic if FAST is to be used to initialise feature tracking as these points will likely not discriminate well. It is better to have at least a small gap between features to ensure their descriptor regions do not overlap. In order to meet this specification Rosten and Drummond proposed a non-maximal suppression step in the FAST detector.

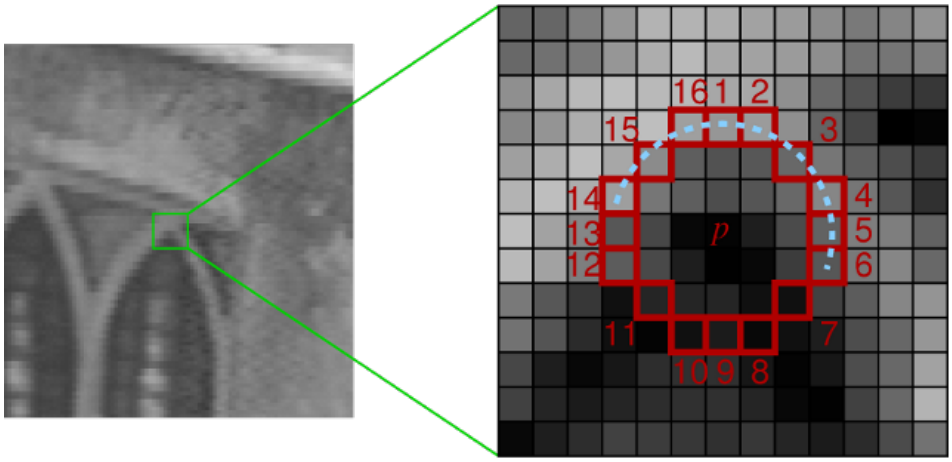


Figure 3.1: The test points on a Bresenham circle of radius 3 are depicted by red boxes. The contiguous pixels used in the fast accelerated segment test around p, with $k = 9$, are shown by the dashed cyan line. (Image source: [43].)

This step calculates a measure of which feature is the most discriminate in a local region, and discards the rest. For a full explanation of this step the reader is referred to the original paper [43].

It should be noted that the detection of FAST features are rotationally invariant, but are not robust to large changes in scale. Furthermore, the nature of the FAST algorithm makes it very difficult to optimise for GPU use as it requires a large number of conditional statements. However, the implementation presented by Rosten and Drummond is very well optimised for CPU usage and takes advantage of caching and branch prediction. The OpenCV version of FAST [6] is based off of this implementation, and is the version we use.

3.1.1.2 Harris Corner Detector

The corner detector developed by Harris and Stephens [12] is one of the oldest corner detection algorithms, yet it is still widely used in many forms. The algorithm builds on the earlier work by Morvac [17], with the main difference being that Harris and Stephens use a first order approximation of the patch shifts to calculate the sum of squared differences (SSD) as a function of shift, rather than directly computing it for selected values.

This approximation of the SSD can be expressed as

$$\begin{aligned} s_{u,v}(x,y) &= \begin{pmatrix} x & y \end{pmatrix} \mathbf{H} \begin{pmatrix} x \\ y \end{pmatrix} \\ &= \begin{pmatrix} x & y \end{pmatrix} \left(\sum_u \sum_v w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \right) \begin{pmatrix} x \\ y \end{pmatrix}, \end{aligned} \tag{3.1.1}$$

where $w(u, v)$ is a Gaussian weighting function over the area of the patch. I_x and I_y represent the partial derivatives of the image I. The summation factor of the approximate SSD is known as the Harris matrix $\mathbf{H}$ and is used to determine if the point (x, y) is a feature point.

It has been shown that eigenvalues are directly related to image curvature [12], and thus if both eigenvalues of $\mathbf{H}$ are large, then the point has high curvature and is classified as a corner. The Harris and Stephens corner score takes into account that the direct computation of eigenvalues is expensive, and thus they suggest the following function:

$$c(\mathbf{H}) = \det(\mathbf{H}) + k[\text{trace}(\mathbf{H})]^2, \tag{3.1.2}$$

where k is a tunable parameter. The effect of image curvature on the eigenvalues can clearly be seen in figure 3.2.

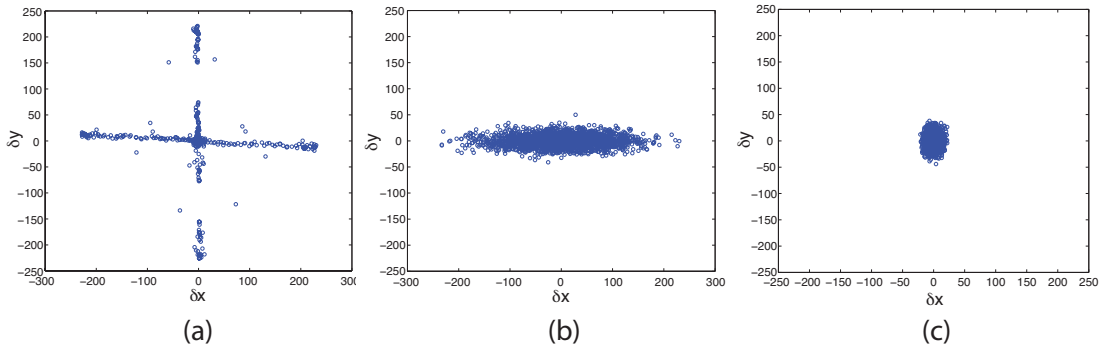


Figure 3.2: The I_x response plotted against the I_y response over three patches we are considering for Harris corner detection: (a) the derivatives when a corner exists, (b) when a line exists; (c) when no corner exists. It can be seen that the corner will produce two large eigenvalues and the texture-less patch none.

While the Harris corner score avoids the computation of eigenvalues in order to speed up computation, Shi and Tomasi [44] propose that the eigenvalues actually be calculated in order to improve the repeatability and robustness of the detected corners. They further propose that a threshold be placed on the smallest eigenvalue, such that if that eigenvalue is larger than the threshold then the point is in fact a corner. The benefits of this corner measure comes from the fact that regions with multiple blobs or non-straight lines can produce eigenvalues which appear to be significant if measured by the way of the Harris corner measure, but prove not to be individually significant when evaluated independently. Such behaviour can lead to false classification by the Harris measure. By ensuring that the smallest eigenvalue is larger than some threshold the number of mis-classifications can be reduced.

As with FAST corners, Harris corners are not robust to large changes in scale but are fairly robust to rotational motion. While Harris corners are more computationally expensive than FAST corners, the nature of the algorithm allows for easy

implementation on a GPU. The GPUImage framework [23], which we use for all the GPU processing, provides such an implementation along with the more robust Shi-Tomasi corner measure.

3.1.2 Feature Description

Once features have been found they are distinguished from one another, and matched to features in other images, using descriptors. These descriptors are constructed using information from the areas around feature points. As with detection, descriptors must ideally be robust to camera motion as well as changes in illumination and scale.

Two main classes of feature descriptors exist, namely histogram of oriented gradients (HOG) descriptors and binary descriptors. HOG based descriptors rely on computing the gradient at each point in the feature patch. These gradients are then quantised and used to build up a histogram of the gradient directions. Two main feature descriptors in this class are SIFT [27] and SURF [3]. Although SURF features speed up the process of creating these histograms by using integral images and coarser quantisation, it is still not fast enough to be used in a mobile SfM pipeline.

Due to their high computational cost we do not consider HOG based descriptors a viable solution for our mobile pose estimation. The rest of this section will investigate the binary feature descriptors which have been shown to provide the best performance and robustness to scale and viewpoint changes, as described by Canclini et al. [9] and Heinly et al. [16].

3.1.2.1 BRIEF

Binary robust independent elementary features (BRIEF) [8] was the first binary descriptor to be published, and is one of the few that can be integrated with any feature detector. The key idea behind BRIEF descriptors is that an $S \times S$ image patch can effectively be described by a small number of pairwise pixel intensity comparisons. BRIEF makes use of these pairwise comparisons to create a binary vector of length d, where d is normally chosen as 128, 256 or 512.

We define the BRIEF pairwise test τ on a patch P as

$$\tau(P, \mathbf{x}, \mathbf{y}) = \begin{cases} 1, & \text{if } P(\mathbf{x}) < P(\mathbf{y}), \\ 0, & \text{otherwise}, \end{cases} \tag{3.1.3}$$

where $P(\mathbf{x})$ is the pixel intensity in a smoothed version of the image at $\mathbf{x} = (u, v)$.

Using the binary test defined in equation 3.1.3 we generate a d-dimensional bitstring as follows:

$$f_d = \sum_{i=1}^{d} 2^{i-1} \tau(P, \mathbf{x}_i, \mathbf{y}_i). \tag{3.1.4}$$

The BRIEF algorithm can use one of multiple strategies in order to generate the d test locations, that is the $(\mathbf{x}_i, \mathbf{y}_i)$-pairs in equation 3.1.4, and a complete overview

of these strategies can be found in the work of Calonder et al. [8]. Based on the results from the paper [8], it is recommended that the components of both test points are sampled from the isotropic Gaussian distribution $\mathcal{N}(0, \frac{1}{25}S^2)$.

BRIEF features do not correct for rotation or scale, but can easily be extended to support these as is done in the BRISK descriptor.

3.1.2.2 BRISK

Binary robust invariant scalable keypoints or BRISK [24] is different from most other binary feature descriptors in that it makes uses of a hand-crafted sampling pattern which is made up of concentric rings, as seen in figure 3.3.

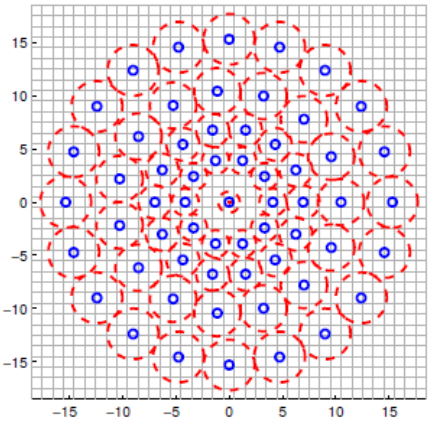


Figure 3.3: The hand-crafted sampling pattern used by the BRISK descriptor. Small blue circles denote the sampling points $\mathbf{p}_i$ and the bigger red circles are drawn with radii corresponding to the Gaussian smoothing kernel used to smooth the intensities at the sampling points. (Image source: [24].)

Unlike BRIEF, the BRISK algorithm also relies on an adapted version of the FAST corner detector. This adapted version works on a scale space, or image pyramid, in order to determine the scale of a feature point. In order for a point to be considered a keypoint, the FAST algorithm detects corners at each level of the pyramid. Non-maximal suppression is applied to each point, starting at the highest level, and a point needs to exhibit the highest corner score with respect to its 8 neighbouring points. Once scale has been determined at each octave of the scale space pyramid a 1-dimensional parabola is fitted to the FAST corner score, and the true scale of the feature is estimated from this regressive fit [24].

When applying the BRISK sampling pattern to a keypoint, a Gaussian smoothing operation is done at each sample point in the patch. The standard deviation of the smoothing filter is made to be proportional to the distance between the keypoint and the sample point. This is done to reduce aliasing effects when sampling the image intensity at particular points in the pattern, as shown in figure 3.3.

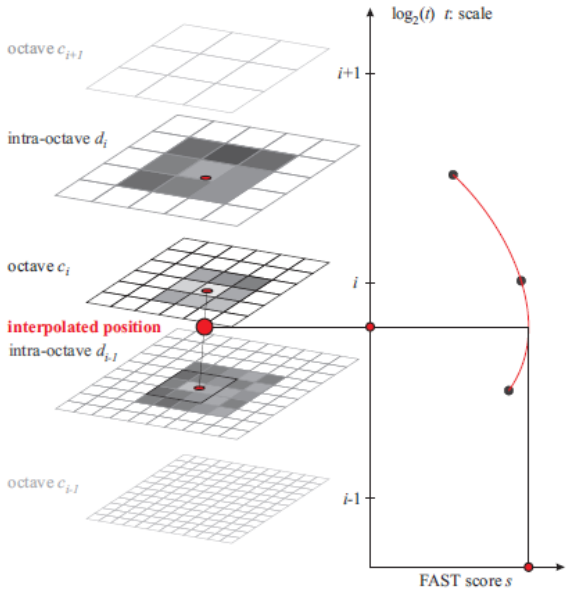


Figure 3.4: The BRISK sampling pyramid. The true estimated scale is read directly from the interpolated FAST scores along a 1D parabola. (Image source: [24].)

BRISK also distinguished between long and short pairs in the sampling pattern. If the distance between a pair of sampling points is less than a certain threshold, the point is referred to as a short pair, otherwise it is classes as a long pair. The BRISK algorithm uses long pairs to estimate the dominant rotation of the feature, while the short pairs are used to generate the binary descriptor. This descriptor is computed in a similar fashion to BRIEF, with the main difference being that the BRISK algorithm compares smoothed pixel intensities for each short pair as opposed to BRIEF which randomly selects pairs for comparison [24].

3.1.3 Feature Matching

Once we have generated descriptors we need to match them in order to determine feature correspondences between frames. One of the biggest advantages of using binary descriptors comes from the fact that matching is orders faster than matching HOG based descriptors like SURF or SIFT. The reason for this is that we are comparing bit-strings on a binary level instead of comparing arrays of floating point numbers.

Binary matching uses the Hamming distance between two descriptors, in place of the Euclidean distance used on HOG type descriptors. The Hamming distance is defined as the sum or the XOR of bits between two bit-strings, in our case feature descriptors, and two points are considered matching if this sum is less than a certain threshold. Assuming we have two feature vectors $\mathbf{a}$ and $\mathbf{b}$ we can formalise the

Hamming distance as

$$d_h = \sum_{i=0}^{n-1} a_i \oplus b_i, \tag{3.1.5}$$

where $\oplus$ represents the bit-wise XOR.

This operation is efficient to perform and is built into most modern CPU SIMD architectures, even on mobile platforms. We can gain further computational performance increases by using lookup tables to return the precomputed sum of bits of the XOR operation. Doing so will result in an operation of O(1) complexity, at the cost of more memory usage. An alternative would be to use a hashing structure on the individual feature vectors so as to only compare a small sub-section of the feature vector, and only compare the rest of the descriptor if the initial comparison is less than a threshold. This branch-and-bound scheme can reduce the memory requirements for the lookup table, while still keeping the algorithm relatively fast.

It is important to note that ambiguous image data will lead to incorrect correspondences, and for this reason feature matching cannot be considered a perfect process. Incorrect feature correspondences pose a significant problem for visual pose estimation and thus they cannot be ignored. We can employ methods to reduce the effect of mismatched features, however these methods come with added computational complexity as we describe in section 4.2.3.

3.2 Feature Tracking

Visual feature tracking refers to the process of tracking detected features across multiple, overlapping frames. They rely solely on image data and do not make use of predefined object models. For these reasons visual feature tracking is well suited to video data in a monocular pose estimation pipeline, where feature points do not move large distances between frames and no prior knowledge of the scene exists.

All visual feature tracking algorithms consist of three main sections:

- feature models;
- motion models;
- matching and update models.

The feature model of a feature tracking algorithm refers to the type of features the algorithm will track. It is generally based on some feature detection algorithm such as FAST or Harris, although many others exist. The motion model is a core aspect of the feature tracking algorithm as it specifies a search area in the new frame for the feature point being tracked. The motion model can be either linear or nonlinear, although the latter is seldom used due to the assumption of small motion that most trackers make. The final stage is the matching and update model. This stage can be either probabilistic or deterministic. Probabilistic models model the

feature location states using a probability density function. The most well known of these methods is the Kalman filter and its extensions. Deterministic models use a measure of similarity to determine the most likely matching point in the new image. This is usually done via a correlation like approach, where a template is extracted from the first frame and searched for in the region specified by the motion model [36].

In order to keep our tracking algorithm as lightweight as possible, we investigate the two main deterministic methods in this section and then propose our own hybrid tracking model based on the feature tracking approach taken by Klein and Murray in their PTAM system [19].

3.2.1 KLT

The Kanade-Lucas-Tomasi (KLT) tracker is one of the most used deterministic feature tracking algorithms. The algorithm is based upon two papers, the first by Lucas and Kanade [29] and the second by Tomasi and Kanade [49]. The work was later unified by Baker and Matthews, who also proposed a more efficient inverse compositional algorithm [2].

The main goal of the KLT algorithm is to align an image template $T(\mathbf{x})$ to an input image $I(\mathbf{x})$. This problem can be formulated as a gradient search over a set of warping parameters $\mathbf{p}$, in increments of $\delta\mathbf{p}$, which minimise the sum of squared difference (SSD) between the two images:

$$e = \sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}, \mathbf{p} + \delta\mathbf{p})) - T(\mathbf{x})]^2, \tag{3.2.1}$$

with respect to $\delta\mathbf{p}$. Here $\mathbf{x} = (x, y)$ is the feature position and $\mathbf{W}(\mathbf{x}, \mathbf{p})$ is the tracking motion model defined by a set of warping parameters $\mathbf{p}$. The warping parameters are updated until the minimisation converges (i.e. $\|\delta\mathbf{p}\| < \epsilon$), as

$$\mathbf{p} \leftarrow \mathbf{p} + \delta\mathbf{p}. \tag{3.2.2}$$

Two main versions of the KLT tracker exist: the forward one given in equation 3.2.1 and an inverse compositional approach. We focus on the inverse compositional algorithm proposed by Baker and Matthews [2], as it is the more efficient formulation of the KLT tracker. The main difference between the inverse compositional algorithm and the original forward algorithm is that the former uses a change of variables to reverse the role of the template and the image. This allows the fixed Hessian matrix of the template T to be used instead of having to recompute the Hessian of the warped image I on every iteration. Due to the fact that the template Hessian is fixed, we can pre-compute it.

The inverse compositional algorithm reformulates the cost function of equation 3.2.1, by switching the roles of the template and the image, as follows:

$$e = \sum_{\mathbf{x}} [T(\mathbf{W}(\mathbf{x}, \delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x}, \mathbf{p}))]^2. \tag{3.2.3}$$

This new cost function is then minimised using a Gauss-Newton gradient descent method. This method first linearises the cost function in equation 3.2.3 using the first order Taylor expansion

$$e = \sum_{\mathbf{x}} [T(\mathbf{W}(\mathbf{x}, \mathbf{0})) + \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \delta \mathbf{p} - I(\mathbf{W}(\mathbf{x}, \mathbf{p}))]^2, \tag{3.2.4}$$

and then iterates to find a local minimum using the partial derivative with respect to $\delta\mathbf{p}$ [2, 18]. In this expression $\nabla T = \left(\frac{\partial T}{\partial \mathbf{x}}, \frac{\partial T}{\partial \mathbf{y}}\right)$ is the gradient of the template.

We obtain a closed form solution for $\delta\mathbf{p}$ at a minimum by taking the partial derivative of equation 3.2.4 with respect to $\delta\mathbf{p}$ and rearranging to get

$$\delta \mathbf{p} = \mathbf{H}^{-1} \sum_{\mathbf{x}} \mathbf{J}^T [I(\mathbf{W}(\mathbf{x}, \mathbf{p})) - T(\mathbf{W}(\mathbf{x}))]^2. \tag{3.2.5}$$

Using the closed form solution for $\delta\mathbf{p}$, the inverse compositional algorithm iteratively updates the currently estimated warp using

$$\mathbf{W}(\mathbf{x}, \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x}, \mathbf{p}) \circ \mathbf{W}(\mathbf{x}, \delta \mathbf{p})^{-1}, \tag{3.2.6}$$

until it converges (i.e. $\|\delta\mathbf{p}\| < \epsilon$). The $\circ$ operator represents the composition of functions. In this formulation $\mathbf{H} = \sum \mathbf{J}^T\mathbf{J}$ is the first order approximation of the Hessian matrix. The steepest descent image $\mathbf{J}$ is defined by

$$\mathbf{J} = \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\bigg|_{\mathbf{p}=0}, \tag{3.2.7}$$

and only changes when the template T is updated.

The inverse compositional algorithm reduces the computational expense of the KLT tracker significantly, however it does little to increase robustness. It can be seen that the KLT tracker relies heavily on the assumption that the starting point $\mathbf{p}_{k+1}^0$ is inside the optimisation region. The standard image based KLT relies on the previously computed warping parameters to initialise the search region in the next frame, and thus exhibits a very strong assumption of small motion.

It was later proposed by Hwangbo et al. [18] that an inertial measurement unit be used to improve the KLT tracker's robustness to larger motion. Their approach deals specifically with pixel motion caused by inter-frame rotations, as this is generally larger than motion due to translation. A more complete description of our take on this approach is dealt with in section 3.3.

3.2.2 Patch Based Tracking

Patch based tracking is similar to the KLT tracker, but uses a more rudimentary search algorithm to find the most probable position of the feature point $\mathbf{p}_{k+1}$ in the next image of the sequence. Instead of a gradient based optimisation search,

as performed in KLT, the patch tracker searches an area with a predefined radius around the position predicted by the motion model. In patch based tracking, the motion model usually assumes the point $\mathbf{p}_{k+1}$ is located at the same position as $\mathbf{p}_k$ [10].

The main goal of patch based tracking is to maximise the correlation score r between the template patch T and the image I over some search area bounded by a fixed pixel radius, such that

$$r = \max_{u,v} \frac{1}{n} \sum_{x,y} \frac{(I(x,y) - \bar{I})(T(x-u, y-v) - \bar{T})}{\sigma_I \sigma_T}, \tag{3.2.8}$$

where $\bar{I}$ and $\bar{T}$ represent the mean of the greyscale pixel intensities for the patch and the template respectively, n is the number of pixels in the template, and σ_I and σ_T are the standard deviations of the same sets of values.

Once the correlation score has been computed for each shift value (u, v) of the template T over the image I, the maximum is chosen in order to obtain the new location of the feature point if its score is greater than some threshold. If the maximum value is below this threshold, the feature point is assumed lost.

The main advantage of patch based tracking comes from the simplicity of its implementation as well as its ability to provide a measure of how accurate the fit is. The correlation score given in equation 3.2.8 is known as zero mean normalised cross correlation, or ZNCC for short, and provides a robust method of measuring similarity between two patches even under large changes in illumination.

Patch based tracking is, however, not robust to large affine warps or rotational motions which place the new feature position $\mathbf{p}_{k+1}$ outside of the search radius. Furthermore, these large motions can skew the visual appearance of a patch to such an extent that the visual similarity between the patches is severely reduced. While the search radius can be increased it is not a robust solution as it can lead to many more false positives, especially in areas of repetitive pattern. A better solution is to use a more accurate motion model, and warp the template patch to approximate the predicted appearance of the image patch in the new image.

While patch based tracking is easy to implement, its computational performance is directly related to the number of features and the size of the search region around each feature. For these reasons it is best implemented in an image pyramid approach where a few coarse features can be tracked over a larger area in order to allow for smaller patches and search radii on lower pyramid levels.

3.3 Hybrid Feature Correspondence

Our hybrid feature tracking algorithm makes use of feature detection as well as a set of algorithms and assumption based on the feature tracking approaches outlined in section 3.2. The algorithm is similar in style to the one proposed by Klein and

Murray in their PTAM system [19], whereby feature points are tracked using detected corners and image patches. We extend this framework by incorporating the GPU of the device and inertial data into the pipeline. Doing so increases robustness to larger inter-frame motions caused by camera rotation and decreases computational complexity by reducing the search space for feature correspondences. A full algorithmic overview of our approach is provided in algorithm 1.

3.3.1 Using Feature Detection to Assist Tracking

As with both the KLT and patch based tracking, we need to determine which points to track. This is usually done using a feature detector such as Harris or FAST. Our hybrid algorithm is initialised and expanded in a similar manner as the other visual feature tracking algorithms we described in section 3.2. We assume that the map has been initialised and that at least two key frames exist in the system already. An explanation of how to fulfil these assumptions is covered in chapter 5.

Tracking initialisation begins by first extracting feature points from the last key frame and adding any new points to the map via a least-squares triangulation method. All the observed map points in that key frame are then added to the initial feature tracking set C and template patches are extracted for each feature. These points will be tracked through all the image frames, until the next key frame is detected, using a motion model $\mathbf{f}$ and a similarity function such as normalised cross-correlation (NCC).

If the similarity score falls below a certain threshold t the feature is marked as lost and the motion model is used to update the feature's location. If the feature has not been recovered after m frames it is removed from the tracking set C altogether. Once a certain percentage of features have been lost, and if the camera pose has changed substantially in terms of rotation or translation, the tracker will begin looking for a suitable frame to be the next key frame. The new key frame is then used to extend the map and increase the size of the tracking set such that tracking can continue until a new key frame is once again required. In section 5.4 we elaborate more on these processes.

The main difference between our proposed tracker and the other trackers we discussed comes from our formulation of the tracking stage and motion model. Instead of relying on feature detection only at key frames, we extract features at every frame in the video sequence. Doing so allows us to reduce the number of checks for inter-frame correspondence from an iterative search of the pixel locations in the area A around the projected feature point $\mathbf{x}_j$, down to a plausible subset of patches $P \subseteq A$. These patches are extracted from neighbouring feature points $\mathbf{x}_i$ detected in the current image. The plausibility set P can be generated with the use of an approximate nearest neighbour algorithm, such that

$$P \leftarrow \mathrm{KNN}(\mathbf{x}_j, C, 20\mathrm{px}), \tag{3.3.1}$$

where KNN is the K nearest neighbour function, $\mathbf{x}_j$ is the feature whose neighbouring points we want to locate, C is the set of all feature points in the image and all

points within 20 pixels of the query point are selected [4]. Figure 3.5 provides a more intuitive visual representation of this approach.

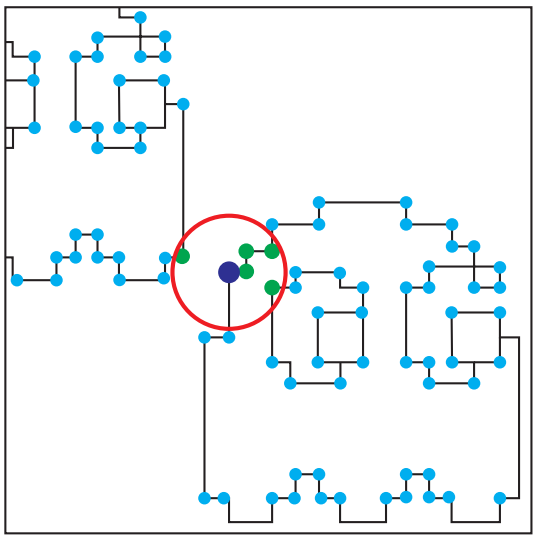


Figure 3.5: We can reduce the search space from all pixels in an area A (red circle) around the projected tracking point $\mathbf{x}_j$ (blue dot), down to a plausible subset P of patches extracted from neighbouring features (green dots) detected in the current frame. The cyan dots represent all the detected features in the current frame which are used in the nearest neighbour search but fall outside of the currently tracked point's neighbourhood.

We can then calculate a similarity score between the warped template patch $T(\mathbf{W}(\mathbf{x}_j))_k$ and the image patch $I(\mathbf{x}_i)_{k+1}$ extracted at each test location $\mathbf{x}_i$ in the plausibility set P. The warping function $\mathbf{W}$ can be determined from the previous frames and is updated only when significant motion has occurred. Furthermore it is also possible to update the template patch when the similarity score falls below a threshold τ, where τ is greater than the feature loss threshold t.

In order to mitigate the cost of running feature detection on every frame we make use of a GPU implementation of the Harris corner detector on the iPhone 5 [23]. For the approximate nearest neighbour search we use the nanoFlann k-d tree implementation which shows improved performance over the FLANN library in the OpenCV toolbox [4]. We can further improve the performance of the k-d tree by initialising it using a pre-sorted set of feature points, where each point is sorted by its x-coordinate and then its y-coordinate. Fortunately for us the GPUImage library inherently supplies the list of feature points in this ordering due to the way it reads them from GPU memory.

Our algorithm reduces the computational complexity of searching for a feature's new location, $\mathbf{x}_{j,k+1}$, in a radius r around the predicted point down to $O(\log(M)+N)$ where N is the number of pixels in the template and M is the total number of feature points in the image. This is in contrast to the $O(nN+n^2)$ per feature per iteration

Algorithm 1 Hybrid Tracking Algorithm

Extract feature points from the current frame I_{k+1}
Generate a k-d tree K from these points
for all features $\mathbf{x}_j$ in the tracking set C **do**
    if lost = true **then**
        $\mathbf{x}_{j,k+1} \leftarrow \mathbf{f}(\mathbf{x}_{j,k})$
    end if
    ▷ Select the K nearest neighbours to the predicted feature location
    $P \leftarrow \text{KNN}(\mathbf{x}_{j,k+1}, C, 20\text{px})$
    $\eta_{\text{min}} \leftarrow \infty$
    for all features $\mathbf{x}_i$ in P **do**
        Update the warping function $\mathbf{W}$ based on some motion model
    ▷ Calculate similarity between the patch and the warped template
        $\eta_{\text{similarity}} \leftarrow \text{NCC}(T(\mathbf{W}(\mathbf{x}_j))_k, I(\mathbf{x}_i)_{k+1})$
        $\eta_{\text{min}} \leftarrow \min(\eta_{\text{min}}, \eta_{\text{similarity}})$
    end for
    if $\eta_{\text{min}} > \tau$ **then**
        Extract new template $T(\mathbf{x}_{\text{min}})$ at the current location
    end if
    if $\eta_{min} > t$ **then**
        $\mathbf{x}_{j,k+1} \leftarrow \mathbf{x}_{\text{min}}$
        lost $\leftarrow$ false
    else if lost for more than m frames **then**
        $C \leftarrow C - \mathbf{x}_j$
    else
        lost $\leftarrow$ true
    end if
end for
$\eta_{\text{ratio}} \leftarrow \frac{C}{C_{k-1}}$
$\delta_{\text{pose}} \leftarrow \text{diff}(\mathbf{P}_{\text{keyframe}}, \mathbf{P}_{k+1})$
    ▷ Determine if a new key frame is needed
if $\eta_{\text{ratio}} < t_{\text{setsize}} || \delta_{\text{pose}} > t_{\text{pose}}$ **then**
    $C_{\text{new}} \leftarrow \text{Harris}(I_{k+1})$
    ▷ Extract new features and append them to the tracking set
    $C \leftarrow C \cup C_{\text{new}}$
end if

complexity of the KLT tracker, where n is the number of warp parameters. The KLT tracker has an additional cost of $O(n^2N+n^3)$ per frame, which is countered by our hybrid algorithm's per frame setup cost of $O(M\log(M))$ which is encountered in building the k-d tree for the nearest neighbour searches.

3.3.2 Using Inertial Sensors to Assist Tracking

One of the biggest weaknesses of our proposed tracking algorithm is that we still rely on the same small motion assumptions as the KLT and patch based tracking approaches. However, unlike the KLT we can expand our search radius with less of a negative consequence as our algorithm does not rely on a first order approximation to define a concave optimisation region. However, in our case increasing the search area does reduce the chances of finding the correct correspondence, especially in regions of low texture or repetitive pattern. Thus, as with the KLT and patch based tracking, a better solution is to improve our motion model to better predict a suitable initial search location $\mathbf{x}^0_{j,k+1}$ such that the probability of the correct correspondence falling within the search area A is increased. This principle is depicted in figure 3.6.

By making the assumption that the largest optical flows in video rate tracking are due to camera rotation, we can make use of the gyroscope to improve the robustness of our tracking system. Our approach follows from the one taken by Hwangbo et al. [18] in their inertially aided KLT implementation.

Suppose we have two sequential video frames, I_k and I_{k+1}, an estimate of the relative inter-frame rotation $\mathbf{R}^{k+1}_k$ between these two frames, as well as a set of corresponding points $[\mathbf{x}_{j,k}, \mathbf{x}_{j,k+1}]$. We can express the re-projection of the corresponding 3D world point $\tilde{X}$ into the two frames as

$$\mathbf{x}_{j,k} = \mathbf{K}\left[I_3|\mathbf{0}\right]X = \mathbf{K}\tilde{X}, \tag{3.3.2}$$

$$\mathbf{x}_{j,k+1} = \mathbf{K}\left[\mathbf{R}^{k+1}_k|\mathbf{t}\right]X = \mathbf{K}\mathbf{R}^{k+1}_k\tilde{X} + \mathbf{K}\mathbf{t}, \tag{3.3.3}$$

where $\mathbf{K}$ is the camera calibration matrix (normalised so that its third row is $[0,0,1]$) and $\mathbf{t}$ is the inter-frame translation in the world co-ordinate system. Details of this type of projection, as well as a formal definition of the calibration matrix, follow in chapter 4.

We rewrite equation 3.3.2 as

$$\mathbf{K}^{-1}\mathbf{x}_{j,k} = \tilde{X}, \tag{3.3.4}$$

and substitute this definition of $\tilde{X}$ into equation 3.3.3 such that

$$\mathbf{x}_{j,k+1} = \mathbf{K}\mathbf{R}^{k+1}_k\mathbf{K}^{-1}\mathbf{x}_{j,k} + \mathbf{K}\mathbf{t}. \tag{3.3.5}$$

Simplifying equation 3.3.5 under the assumption that $\mathbf{t}$ is small, we obtain

$$\mathbf{x}_{j,k+1} = \mathbf{H}\mathbf{x}_{j,k}, \tag{3.3.6}$$

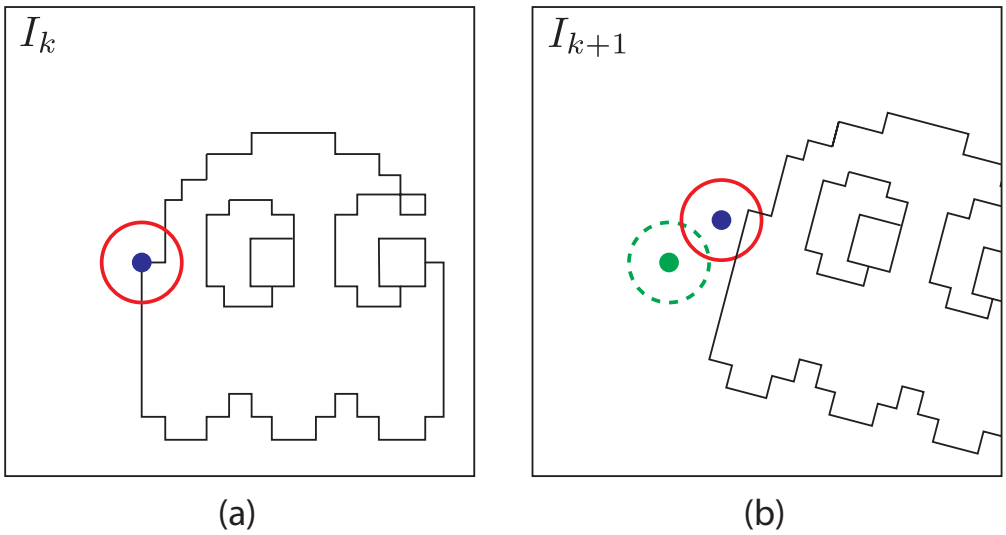


Figure 3.6: Even a small device rotation between two frames I_k and I_{k+1} can lead to large feature movement which can lead to feature loss in conventional tracking techniques such as the KLT. (a) The blue point is the feature point we are tracking, and the red circle indicates the search area A. (b) Using the normal KLT motion model we find that A (dashed green circle) no longer captures the new feature location. Using our improved motion model to predict a more suitable initial search location $\mathbf{x}^0_{j,k+1}$ (blue point) we enable the actual feature location to be captured by A (red circle).

where

$$\mathbf{H} = \mathbf{K}\mathbf{R}^{k+1}_k\mathbf{K}^{-1}. \tag{3.3.7}$$

It is important to note that equation 3.3.6 holds true only under the assumption that $\mathbf{t}$ is small, which is also an assumption of other tracking methods and thus should not be considered detrimental to tracking.

In order to obtain $\mathbf{R}^{k+1}_k$ we make use of the instantaneous angular velocity $\boldsymbol{\omega} = (p, q, r)^T$ of the device, sampled in-between the capture of the two frames. We can express the equivalent quaternion rate of change directly from the angular body rates as

$$\dot{\mathbf{q}} = \frac{1}{2}\Omega(\boldsymbol{\omega}) = \frac{1}{2}\begin{bmatrix} 0 & r & -q & p \\ -r & 0 & p & q \\ q & -p & 0 & r \\ -p & -q & -r & 0 \end{bmatrix}\mathbf{q}_t, \tag{3.3.8}$$

where $\mathbf{q}_t$ is the initial quaternion position. Using numerical integration techniques with proper quaternion normalisation we can determine the relative rotation between the two frames. This quaternion representation can then be converted to its rotation matrix form by applying equation 2.1.17.

We use $\mathbf{q}_t = \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}^T$ if we are integrating between two sequential frames or we set $\mathbf{q}_t$ to the estimated rotation of the previous frame ($\mathbf{q}_{t-1}$) if tracking was lost in the previous frame. The use of quaternions allows for fast integration of $\boldsymbol{\omega}$

to the rotation $\mathbf{q}$. As the gyroscope values are only integrated over a short period of time the drift on the sensor has little effect and can be ignored.

Using equations 3.3.8 and 3.3.6 we can estimate a more accurate initial search point for each feature in the tracking set C. Furthermore, if a feature is lost in frame I_{k+1} we can continue updating its estimated position using equation 3.3.6 until such a time as the point is found again or scrapped from C due to being lost for too long.

From equation 3.3.6 it is clear that if the inter-frame motion is purely rotational, we can express all feature motion by a single 2D homography $\mathbf{H}$. This homography is a higher order deformation than an affine warp and thus can either be used directly to warp the patch or we can extract an affine warp from $\mathbf{H}$ as was done by Hwangbo et al. [18].

With the knowledge of how we make use of the inertial data to improve the tracking initialisation we refer the reader back to figure 3.6. The green point in I_{k+1} is the initialisation used by the KLT tracker (the previous known location of the feature). It is clear that the rotational motion has placed the new feature location outside of this search area. Using the inertial information from the gyroscope we can minimise the effect of the rotation. This is seen in I_{k+1} where our initial search location (blue point) is closer to the actual feature location and thus the search area A (red circle) now contains the location of the feature point we are tracking.

In this chapter we discussed two approaches to determining feature correspondence, namely feature detection and matching by means of descriptors, and feature tracking by minimising the SSD between a template patch and an image patch. We extended these ideas by presenting a hybrid feature tracking algorithm which makes use of both the previously discussed techniques, and further exploits the nature of video data. The feature correspondences found by this algorithm form the input into the visual pose estimation stage which is discussed in the next chapter.

Chapter 4

Visual Pose Estimation

Visual pose estimation refers to the process of determining a camera's position and orientation relative to some initial position, in our case the first frame of the video sequence.

Two main approaches exist to solving this problem, namely filtering and factorisation. The factorisation approach provides better accuracy than filtering, but this often comes with an increased computational complexity. Filtering methods often make use of Kalman filtering techniques and thus are sensitive to incorrect feature correspondences. If only a few incorrect correspondences are provided it can cause the filter to diverge. Furthermore filtering approaches require that the motion can be represented by a process model, and in the case of full 6DoF motion this model will need to be very general. Factorisation approaches do not rely on a model but rather directly on the measurements provided. While this is also susceptible to divergence due to incorrect correspondences, there are methods to reduce the effect of such errors on the optimisation without needing to rely on a prediction model.

In this chapter we explore the use of factorisation and optimisation algorithms for visual pose estimation, and show how the inclusion of inertial sensor information can increase the robustness as well as reduce the computational load of these algorithms. Furthermore, we propose a reduced version of the bundle adjustment optimisation algorithm which lessens the computational load and increases the feasibility of frame rate pose estimation.

4.1 Projective Geometry

4.1.1 Pinhole Camera Model

Before we can predict a camera's pose based on visual features, we need to define a mathematical model of how a camera forms an image based on light travelling from objects in the world. This model is then used as the basis for determining how we can back-project image features in order to determine their 3D locations relative to

the camera, and thus the camera's pose relative to the objects in the world. We use the pinhole camera model, as described by Hartley and Zisserman [14].

The camera coordinate system is defined as having its origin at the camera's centre of projection, **c**, and its xy-plane parallel to the image plane. The point where Z_c (the z-axis of the camera coordinate system) pierces the image plane is known as the principal point **p**. It is usually close to the centre of the image and defines the origin of the image plane. The principal point is located at a distance f along Z_c, where f is the focal length of the camera. A visual representation of these concepts can be seen in figure 4.1.

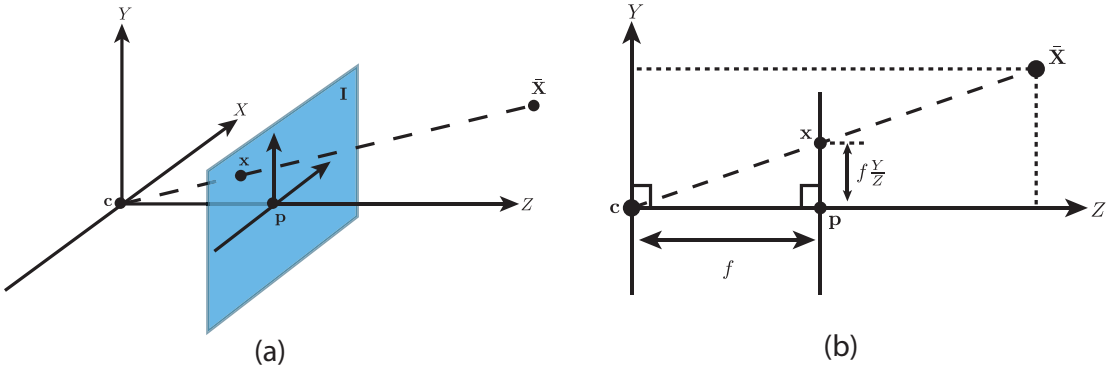


Figure 4.1: The pinhole camera model showing the projection of a world point $\bar{X}$ onto the image plane and axis alignment, creating (a) the camera coordinate system, and (b) the similar triangles relationship from which equation 4.1.1 is derived.

Let $\mathbf{x} = (x, y)^T$ be the projection of a 3D scene point $\bar{X}_c = \left(X, Y, Z\right)^T$ in the camera coordinate system onto the image plane, with the centre of projection at **c**. Using similar triangles we can express this projection as

$$\mathbf{x} = \left(f\frac{X}{Z} + p_x, \quad f\frac{Y}{Z} + p_y\right)^T. \tag{4.1.1}$$

It is important to note that the origin of the image coordinate system is seldom at the same point as **p**, but rather normally at the top left corner of the image. We can define the principal point's location in the image coordinate system as $\mathbf{p} = \left(p_x, p_y\right)^T$, where p_x and p_y represent an offset in pixels between the origin of the image coordinate system and the origin of the image plane. These offsets allow us to move the image plane's origin to coincide with a more convenient point such as the start of our image data structure.

For simplicity we can formulate the nonlinear transform in equation 4.1.1 as a linear expression by using homogeneous coordinates. When using homogeneous coordinates to describe a point on a plane, we use three parameters instead of two. Thus $\mathbf{x} = (x, y)^T$ becomes $\left(x, \quad y, \quad 1\right)^T$, and we denote these homogeneous image coordinates as $\tilde{\mathbf{x}}$. It should further be noted that to go from any homogeneous representation back to 2D image coordinates, we normalise so that the third element is 1.

Indeed, in homogeneous coordinates $\tilde{\mathbf{x}} = k\tilde{\mathbf{x}}$ for any $k \neq 0$. Thus two homogeneous vectors can only be considered equal up to some scale factor. Using homogeneous coordinates we rewrite equation 4.1.1 as

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}, \tag{4.1.2}$$

or more concisely as

$$\tilde{\mathbf{x}} = [\mathbf{K}|\mathbf{0}]X_c, \tag{4.1.3}$$

where $\mathbf{K}$ is known as the camera calibration matrix or intrinsic camera matrix and X_c is the homogeneous version of $\bar{X}_c$.

In addition to correcting for the image origin we can also allow the camera calibration matrix to make corrections for non-square pixels as well as for skew on the imaging sensor. With these added parameters the final camera calibration matrix becomes

$$\mathbf{K} = \begin{bmatrix} \alpha_x f & s & p_x \\ 0 & \alpha_y f & p_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{4.1.4}$$

where α_x and α_y are scaling factors to compensate for non-square pixels, and s allows for slight skew in the imaging sensor.

In general we will have some fixed world coordinate system in which objects are to be reconstructed, and we will want to describe the camera's pose relative to that world coordinate system. Equation 4.1.2 defines a mapping between the camera coordinate system and the image coordinate system, thus we only need to define a transformation from a point $\bar{X}_w$ in world coordinates to the equivalent point $\bar{X}_c$ in the camera coordinate system.

We can define this transformation by a rotation $\mathbf{R}$ from the world frame to the camera frame and a translation $\mathbf{t}$ such that

$$\bar{X}_c = \mathbf{R}(\bar{X}_w - \mathbf{c}) = \mathbf{R}\bar{X}_w + \mathbf{t}. \tag{4.1.5}$$

Thus we can now define the projection of a point in world coordinates onto the image plane with the following simple equation in homogeneous coordinates:

$$\tilde{\mathbf{x}} = \mathbf{P}X_w, \tag{4.1.6}$$

where

$$\mathbf{P} = \mathbf{K}\mathbf{R}[\mathbf{I}|-\mathbf{c}] = \mathbf{K}[\mathbf{R}|\mathbf{t}]. \tag{4.1.7}$$

The matrix $[\mathbf{R}|\mathbf{t}]$ is referred to as the extrinsic camera parameters or the camera pose. The matrix $\mathbf{P}$ is known as the camera projection matrix, or just the camera matrix. The goal of SfM algorithms is to find both the camera matrix $\mathbf{P}$ and the 3D point coordinates $\bar{X}_w$, given image points of such a feature in various frames.

4.1.2 Camera Calibration

As we are working with a single camera, with fixed intrinsic parameters, we can simplify the camera matrix in equation 4.1.7 by removing the calibration matrix. This can be done by pre-computing $\mathbf{K}$ using a camera calibration procedure. We use Zhang's camera calibration procedure [57, 14], and the camera calibration toolbox for Matlab [5].

The camera calibration procedure makes use of a checker-board pattern with known square sizes. The camera is used to take photos of this checker-board from various angles and distances and then these images and the known data are fed into the calibration procedure. Examples of the images we used for calibration are shown in figure 4.2. From this point related features (corners of the pattern) are selected and then optimised to sub-pixel accuracy. Once we have corresponding points between the set of calibration images we can calculate, among other things, the calibration matrix $\mathbf{K}$ common to all the images.

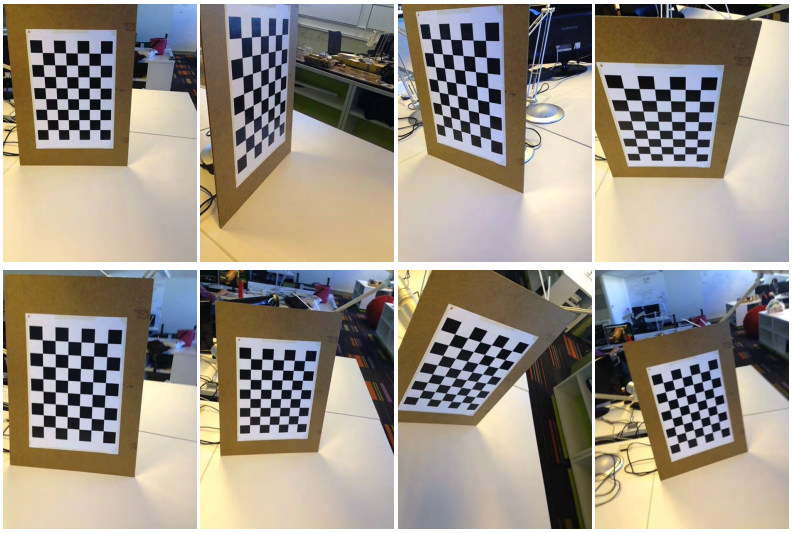


Figure 4.2: Examples of our camera calibration images. The squares on our checker-board pattern are $40mm \times 40mm$ and the board was kept stationary while the camera was moved to various positions.

From this procedure we found the intrinsic camera parameters of the iPhone 5 to be

$$\mathbf{K} = \begin{bmatrix} 589.56 & 0.00 & 241.00 \\ 0 & 589.00 & 312.78 \\ 0 & 0 & 1 \end{bmatrix}. \tag{4.1.8}$$

These values for $\mathbf{K}$ was found to be relatively consistent across multiple iPhone 5 devices.

Having knowledge of the camera calibration matrix allows us to simplify our formulation of the camera matrix in equation 4.1.7 to $\mathbf{P} = [\mathbf{R}|\mathbf{t}]$ on condition that we

work with normalised image coordinates. We denote normalised image coordinates with a hat, and define them as

$$\hat{\mathbf{x}} = \mathbf{K}^{-1}\tilde{\mathbf{x}}. \tag{4.1.9}$$

For a complete description of Zhang's algorithm the reader is referred to the original paper [57].

4.2 Epipolar Geometry and the Essential Matrix

In this section we consider the case where a point in world coordinates is captured by two cameras with the same camera matrix. It is important to note that multiple cameras with the same camera matrix can also be thought of as a single camera at two different instances in time, as is the case in our system. From this point onwards the term camera refers to a video frame captured at a specific instance of time, and not to another physical camera. We consider the case of two cameras as it is the minimal number of views required in order to recover 3D information from image correspondences. Furthermore we use pairs of key frames to initialise and extend a map of the recovered structure and thus the two view case is essential to our application.

Consider the case where a 3D world point $\bar{X}$ is captured by two cameras, with their respective centres at $\mathbf{c}$ and $\mathbf{c}'$ in world coordinates, such that the corresponding image points are $\mathbf{x}$ in the first image and $\mathbf{x}'$ in the second. Figure 4.3 clarifies. The line joining the two camera centres is known as the baseline of the two-camera setup. Furthermore, there is a plane formed by $\mathbf{c}$, $\bar{X}$ and $\mathbf{c}'$ which is known as an epipolar plane.

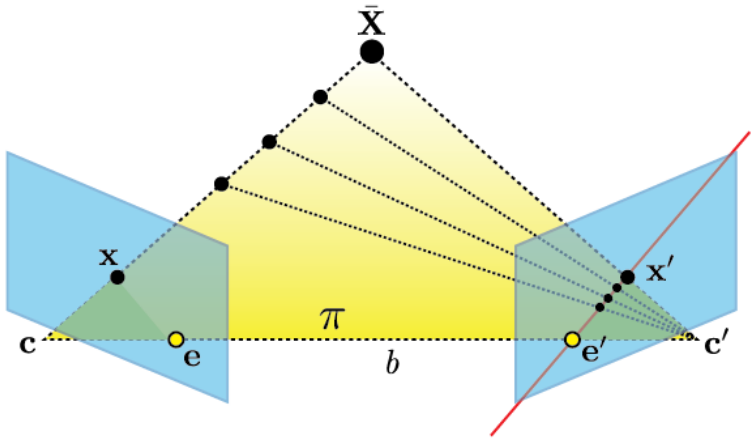


Figure 4.3: A visual representation of the epipolar geometry constraints. It can be seen that when the image point $\mathbf{x}$ is transformed into the second image, its position is known to be along the epipolar line (red line) in that image plane.

Suppose we only know $\mathbf{x}$. It is clear from figure 4.3 that $\bar{X}$ will then fall on the line through $\mathbf{c}$ and $\mathbf{x}$. Thus the corresponding point $\mathbf{x}'$ will fall on the line created where the epipolar plane intersects the second image plane. This line is known as an epipolar line. The property that the correspondence of a feature in the one image is restricted to a line in the other image is generally referred to as the epipolar constraint, and it is useful in that it constrains the correspondence search space to a single line, rather than the whole image. It should be noted that all epipolar lines in an image intersect at a single point, called the epipole of that image, seen as $\mathbf{e}$ and $\mathbf{e}'$ in figure 4.3. The epipole is also the point at which the baseline intersects the image plane [14, 47].

Using the properties of epipolar geometry and our knowledge of the camera calibration matrix $\mathbf{K}$ it can be shown that there exists a 3×3 homogeneous matrix $\mathbf{E}$, called the essential matrix, which satisfies

$$\hat{\mathbf{x}}'^{T}\mathbf{E}\hat{\mathbf{x}} = 0, \tag{4.2.1}$$

for any pair of normalised image points $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}}'$ which correspond to the same 3D feature [47]. The essential matrix has rank 2 and five degrees of freedom. The property described in equation 4.2.1 is derived from the epipolar constraint between a pair of corresponding points. Equation 4.2.1 provides a measure of how closely the detected point $\mathbf{x}$ in the first image lies to the epipolar line generated by the transformation of its corresponding point in the second image, and vice versa for the point $\mathbf{x}'$ in the second image. This can be convenient for detecting wrongly corresponded features.

The essential matrix is directly related to the pose of the second camera relative to the first, such that

$$\mathbf{E} = [\mathbf{t}]_{\times}\mathbf{R}, \tag{4.2.2}$$

where $[\mathbf{t}]_{\times}$ is the skew-symmetric cross product matrix of $\mathbf{t}$, that is $[\mathbf{t}]_{\times}\mathbf{b} = \mathbf{t} \times \mathbf{b}$ for any $\mathbf{b}$.

Once the essential matrix has been determined we can decompose it to obtain the camera matrix $\mathbf{P}$ and thus the rotation $\mathbf{R}$ and translation $\mathbf{t}$. This decomposition is not trivial as it produces four geometrically correct solutions, of which only one is the true solution. We can test which solution is correct by triangulating the observed points with each solution. The rotation and translation which triangulate to a position in front of both cameras is deemed as the correct solution [14, 47]. The four possible solutions are depicted in figure 4.4.

Thus once we have determined $\mathbf{E}$ we have determined all the parts of equation 4.1.7 and thus can recover the pose of the second camera relative to the first, as well as the original 3D point $\bar{X}$ via the process of triangulation (section 4.2.1). Furthermore, we can use the obtained rotation and translation to augment the user's view of the world with computer generated objects.

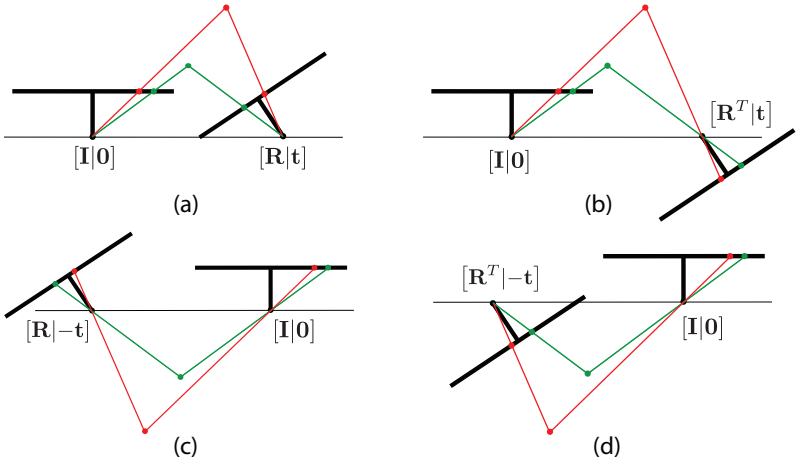


Figure 4.4: The essential matrix decomposition produces four possible solutions which all satisfy the re-projection properties of the essential matrix: (a) the correct solution with all points in front of both cameras; (b) the second camera has been rotated by the inverse rotation; (c) the second camera has been translated in the opposite direction; (d) the second camera has been rotated by the inverse rotation and translated in the negative direction.

4.2.1 Triangulation

In order to recover the 3D structure $\bar{X}$ from a pair of 2D point correspondences $(\mathbf{x}, \mathbf{x}')$ and the essential matrix we make use of the linear least-squares method presented by Hartley and Zisserman [14].

Given the camera matrices for two images $\mathbf{P}$ and $\mathbf{P}'$, which can be obtained by decomposition of $\mathbf{E}$, we can calculate $\bar{X}$ such that $\mathbf{x} = \mathbf{P}\bar{X}$ and $\mathbf{x}' = \mathbf{P}'\bar{X}$. As these equations are homogenous we first need to eliminate the unknown scale factor This is done by rather considering those two equations in the form $\mathbf{x} \times \mathbf{P}\bar{X} = \mathbf{0}$ and $\mathbf{x}' \times \mathbf{P}'\bar{X} = \mathbf{0}$. Considering the first of these, we obtain the following set of linear equations

$$\begin{aligned} x\mathbf{p}_3^T\bar{X} - \mathbf{p}_1^T\bar{X} &= 0 \\ y\mathbf{p}_3^T\bar{X} - \mathbf{p}_2^T\bar{X} &= 0 \\ x\mathbf{p}_2^T\bar{X} - y\mathbf{p}_1^T\bar{X} &= 0 \end{aligned} \tag{4.2.3}$$

where $\mathbf{p}_j^T$ is the j-th row of $\mathbf{P}$ and $\mathbf{x} = (x, y, z)^T$.

By expanding the second point's constraint in the same way we can combine the first two equations from each set (equation 4.2.3) to obtain

$$\mathbf{A}\bar{X} = \begin{bmatrix} x\mathbf{p}_3^T - \mathbf{p}_1^T \\ y\mathbf{p}_3^T - \mathbf{p}_2^T \\ x'\mathbf{p}'^T_3 - \mathbf{p}'^T_1 \\ y'\mathbf{p}'^T_3 - \mathbf{p}'^T_2 \end{bmatrix} \bar{X} = \mathbf{0}. \tag{4.2.4}$$

Equation 4.2.4 is linear in $\bar{X}$ and can thus be solved by finding the right nullspace of $\mathbf{A}$. This nullspace can be interpreted as the least-squares approximation of the 3D point $\bar{X}$ (the two back-projected rays do not necessarily intersect, due to inaccurate feature detection, therefore it might not be possible to find a nontrivial solution for $\bar{X}$ exactly).

4.2.2 The 8-point Algorithm

Although the essential matrix has five degrees of freedom, and can be solved by knowing only 5 correspondences between the two images, this solution is not easy to formulate and requires nonlinear techniques. An easier approach is to use a minimum of 8 normalised point correspondences. This allows us to formulate a linear least-squares solution for the essential matrix. The method is known as the 8-point algorithm [13, 14, 47].

The approach taken by the 8-point algorithm follows on directly from equation 4.2.1. By performing the matrix multiplication we can express equation 4.2.1 as the inner product

$$\begin{pmatrix} \hat{x}'\hat{x} & \hat{x}'\hat{y} & \hat{x}'\hat{z} & \hat{y}'\hat{x} & \hat{y}'\hat{y} & \hat{y}'\hat{z} & \hat{z}'\hat{x} & \hat{z}'\hat{y} & 1 \end{pmatrix} \cdot \mathbf{e} = 0, \tag{4.2.5}$$

where $\mathbf{e} = \begin{pmatrix} e_{11} & e_{12} & e_{13} & e_{21} & e_{22} & e_{23} & e_{31} & e_{32} & e_{33} \end{pmatrix}^T$ is the vectorised version of the essential matrix, such that e_{ij} is the element at row i and column j. The elements of the row vector in equation 4.2.5 are formed from the normalised homogeneous image coordinates of a pair of corresponding points $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}'}$, as described by equation 4.1.9.

Equation 4.2.5 provides one constraint on $\mathbf{E}$, thus by stacking at least 8 such equations (each corresponding to a different pair of matching features) into a matrix $\mathbf{A}$ we can solve for the vector $\mathbf{e}$ and thus for the essential matrix. Since the elements of $\mathbf{e}$ can only be solved up to scale, we further solve equation 4.2.5 with the constraint that $\|\mathbf{e}\| = 1$.

The solution to equation 4.2.5 can be found easily by taking the singular value decomposition (SVD) $\mathbf{A} = \mathbf{USV}^T$ and setting $\mathbf{e}$ as the last column of $\mathbf{V}$. This vector $\mathbf{e}$ can then be reshaped into the 3×3 essential matrix [14, 40]. Note that if the system does not have a nontrivial solution (due to inaccurate feature coordinates for example) this SVD approach will give a least-squares approximation, similar to the case of triangulation.

4.2.3 RANSAC

While it is easy to compute the essential matrix with 8 point correspondences, there is a strong possibility that the solution will be a suboptimal one. The reason for this is due to the high likelihood of incorrect feature correspondences. If we happen to include an incorrect correspondence in the computation of the essential matrix, the

solution is likely to be incorrect. The reason for this is that least-squares methods are significantly influenced by outlying data.

Furthermore, the 8-point algorithm makes the assumption that the feature points are not all on a single plane (for sufficient linear independence in the rows of $\mathbf{A}$). Thus if our chosen 8 point correspondences happen to coincide with 3D features on a similar plane, in the world, an ambiguity is produced as the problem is under-constrained [14, 13].

In order to increase the probability of finding an optimal solution for $\mathbf{E}$ we employ a method known as random sample consensus (RANSAC). It is an iterative approach, which solves for the essential matrix by randomly selecting 8 point correspondences from the set of all point correspondences. The accuracy of such a solution is then determined by applying it to the set of all point correspondences, and measuring how many point correspondences agree with the solution. The set of corresponding points which agree with the solution of $\mathbf{E}$ is known as the consensus set. This process is then repeated for a multiple iterations (1000 in our case) and the essential matrix corresponding to the largest consensus set is taken to be the correct solution [58, 14].

By finding the largest consensus set we also end up with a plausible set of inlying points. The motivation behind RANSAC is that there is typically no structure in outlying or incorrect point correspondences and thus there is a higher probability that the consensus set consists only of inliers. It should be noted that due to the assumption of structure only existing amongst correct correspondences, the final consensus set can be significantly smaller than the set of outlying points.

In order to measure how many point correspondences agree with the solution we make use of the Sampson distance instead of directly applying equation 4.2.1. The Sampson distance is the first order approximation to the geometric error, which calculates the error between the observed points and the closest points which satisfy the epipolar constraints. We define the Sampson distance for a set of n point correspondences as

$$e = \sum_{j=1}^{n} \frac{(\hat{\mathbf{x}'}_j^T \mathbf{E} \hat{\mathbf{x}}_j)^2}{(\mathbf{E}\hat{\mathbf{x}}_j)_1^2 + (\mathbf{E}\hat{\mathbf{x}}_j)_2^2 + (\mathbf{E}^T\hat{\mathbf{x}'}_j)_1^2 + (\mathbf{E}^T\hat{\mathbf{x}'}_j)_2^2}, \tag{4.2.6}$$

where $(\mathbf{E}\hat{\mathbf{x}}_j)_i^2$ represents the square of the i-th element of the vector $\mathbf{E}\hat{\mathbf{x}}_j$.

By using RANSAC along with the Sampson distance we can optimally find a solution to both the camera pose as well as the 3D location of the observed points. However, the 8-point algorithm coupled with RANSAC is slow to solve, as RANSAC usually requires between 500 and 1000 iterations in order to provide 95% confidence that the solution set consists of only inlying point correspondences [58].

4.3 Bundle Adjustment

While other SfM techniques, such as the 8-point algorithm or the perspective-n-point method [47], can be performed sequentially with triangulation in order to determine new 3D points as well as the camera pose of a new frame, they can suffer from significant error accumulation. By performing optimisation over all points and all camera poses, it is possible to significantly reduce this accumulated error.

Bundle adjustment (BA) refers to an optimisation technique used in order to refine both camera pose and 3D structure. Bundle adjustment minimises the re-projection error between a detected feature point and a re-projection of the corresponding 3D point onto the image plane. This problem can be represented by the sum of squares of a large number of nonlinear, real valued functions, and thus can be solved using a nonlinear least-squares approach.

A general approach to solving the bundle adjustment problem is by using the Levenberg-Marquardt (LM) optimisation algorithm [31, 41], which has become a popular choice due to its ease of implementation and its ability to converge from a wide range of initialisations. The LM algorithm behaves like Guass-Newton optimisation when it is far from the solution and changes its behaviour to represent a steepest descent approach as it gets nearer to the optimal solution.

Apart from using a fast nonlinear least-squares solver, such as LM, many other approaches have been taken to further decrease the computational load of bundle adjustment. The most common of these exploits the sparsity of the approximate Hessian matrix. In order to use the LM algorithm the Jacobian of the cost function with respect to the camera parameters and 3D feature positions needs to be evaluated. This leads to a sparse Jacobian structure as can be seen in figure 4.5. This sparse Jacobian structure in turn leads to a sparse approximate Hessian matrix [15, 50]. This sparsity is caused by a lack of interaction among the parameters related to different cameras and 3D points. That is, not all points are observed by all cameras.

As bundle adjustment is formulated as a local optimisation problem, it requires a good initialisation in order to converge. For this reason it is normally used as a final step in the general SfM pipeline after some initial pose estimation and reconstruction by, for example, the 8-point algorithm and triangulation.

In the rest of this section we propose an alternative formulation of bundle adjustment which reduces the problem from a global batch optimisation into a local, incremental optimisation. We further demonstrate how the use of sensors and the knowledge of previous frames and key frames (specific frames which have a significant baseline and are used to reconstruct 3D world points) can be used to speed up the convergence of bundle adjustment. Using these two techniques we show how we can use bundle adjustment to localise the camera at frame rate, thus removing the need for a two stage localisation approach as seen in most other SfM pipelines [19].

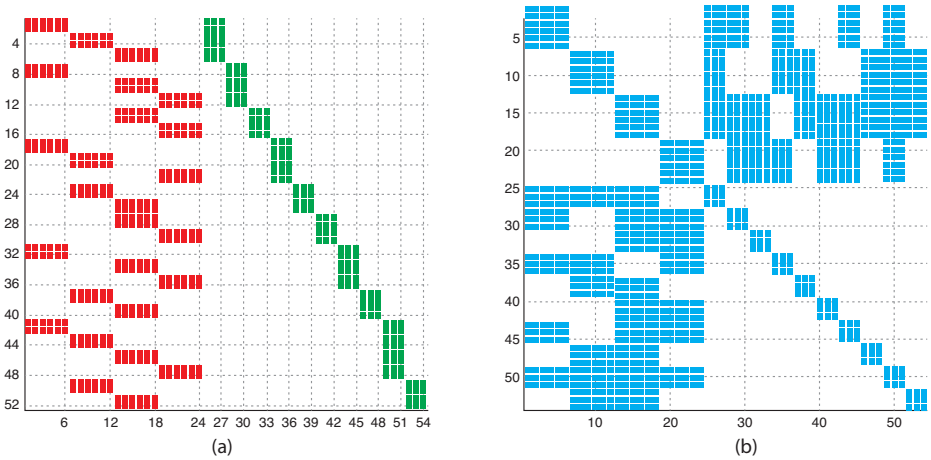


Figure 4.5: The sparse block structure of the Jacobian and the approximate Hessian matrix for a toy problem with four cameras and ten 3D points. The elements corresponding to the camera parameters in the Jacobian are shown in red, and the 3D points in green.

4.3.1 Global Bundle Adjustment

Global bundle adjustment (GBA) refers to the use of all camera, feature correspondence and 3D point information in order to optimise the camera poses and 3D point positions over the set of all frames. This optimisation is large in dimension and can take anything from a few seconds to hours depending on the size of the problem (the number of cameras and the number of points). The reader is reminded that in the case of a monocular system, such as ours, cameras refer to image frames captured at different instances of time with a single camera.

Consider a system where J cameras, with camera matrices $\mathbf{P}_j, j = 1, \ldots, J$, observe a static scene of K 3D world points $X_k, k = 1, \ldots, K$. Let $\mathbf{x}_{j,k}$ be an observation (from the feature tracker) of 3D point k in the j-th frame. We can then represent the cost function to be minimised for a single such image point as

$$\|\mathbf{x}_{j,k} - \tilde{\mathbf{x}}_{j,k}\|, \tag{4.3.1}$$

where for the sake of brevity we represent the projection of a 3D point X_k onto the j-th image frame as $\tilde{\mathbf{x}}_{j,k} = \text{Proj}(\mathbf{P}_j, X_k)$. This projection is performed using equation 4.1.7 and the j-th camera matrix $\mathbf{P}_j$, and transforming from homogeneous coordinates back to 2D image coordinates. It should be noted that only the extrinsic camera parameters $\mathbf{R}_j$ and $\mathbf{t}_j$ are included in the optimisation as $\mathbf{K}$ is constant in the case of a single camera.

Due to occlusions, limited field of view and bad feature correspondence, not all 3D points are visible in all the frames. For this reason we define a subset V_j which represents only the features visible in the j-th frame. Using equation 4.3.1 and the

subset V_j we can define the GBA least-squares cost function to be

$$e(\mathbf{y}) = \sum_{j=1}^{J} \sum_{k \in V_j} \|\mathbf{x}_{j,k} - \tilde{\mathbf{x}}_{j,k}\|^2, \tag{4.3.2}$$

where $\mathbf{y}$ is a vector containing all the parameters of the cameras and 3D points [50, 15]. Bundle adjustment is a local optimisation procedure and thus requires reasonably good initial estimates of the extrinsic camera parameters as well as of the 3D feature positions in order to improve the chances of convergence to the correct, globally optimal solution.

In the PTAM system [19], global bundle adjustment is run periodically in order to optimise all the key frames and map points. This optimisation is done when the camera is not exploring, as the computational load can then be handled by the processor. Furthermore, Klein and Murray exploit the sparsity of the GBA problem in order to reduce the computational complexity from $O((N+K)^3)$ to $O(N^3)$ where K is the number of map points and N is the number of key frames [19]. In our system we do not use global bundle adjustment, and make the argument that a well initialised local bundle adjustment will (operating on only a small number of key frames) result in a near optimal pose estimate and reconstruction.

4.3.2 Local Bundle Adjustment

In order to reduce drift in the camera pose estimates of key frames, we need to make use of bundle adjustment. However, the large scale optimisation approach taken by GBA is far too computationally intensive to make it feasible for real-time use on a mobile device.

The approach taken by Klein and Murray to make bundle adjustment feasible on a mobile device is to limit the maximum number of 3D points stored in the map. This means discarding key frames and map features [20]. While this approach performs well for tracking applications, it can reduce the robustness of the system as re-localisation and camera pose estimation rely on previous key frames and 3D map points.

Although most modern mobile devices have a multicore CPU, running full GBA in a separate thread still puts significant pressure on the memory and CPU caches, such that it remains an infeasible approach on most modern mobile devices. For these reasons we propose an alternative solution. By reducing the size of the bundle adjustment problem we can increase the feasibility of its use on a mobile device.

In order to reduce the number of parameters in the bundle adjustment we use a windowed approach, whereby only the previous three key frames are used in the optimisation. We define the cost function in equation 4.3.2 over these three frames only. We then fix the pose of the first camera and allow only the pose of the second and third cameras to be updated. We allow all the 3D points which are visible in more than one of the cameras in the bundle to be updated while fixing the locations

of those points visible only in a single camera. A visual representation of this approach is depicted in figure 4.6.

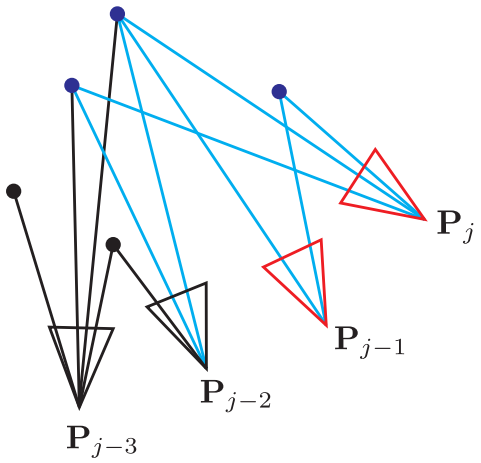


Figure 4.6: The cost function in equation 4.3.2 is defined over cameras $\mathbf{P}_j$, $\mathbf{P}_{j-1}$ and $\mathbf{P}_{j-2}$. The pose of the cameras in red can be updated, as can the 3D points indicated in blue. Cameras and 3D points marked in black are fixed in order to preserve consistency with previous bundles.

The reason behind fixing the location of the first camera and only including features observed in at least two frames is to provide consistency across various windows of key frames. By doing this we are attempting to prevent a case whereby the oldest key frame in the bundle is moved significantly in order to correct for a poor quality new key frame. This case would cause previous bundles to become inaccurate and introduce drift into our optimisation. It should be noted that this local bundle adjustment (LBA) is still susceptible to drift as we do not optimise over all the cameras and 3D points. However, the drift is significantly reduced when compared to the drift which occurs when applying the 8-point algorithm directly without bundle adjustment.

Our local bundle adjustment is run every time a new key frame is added to the system. The optimisation itself does not make use of inertial sensor information as the key frames have already been localised using an accurate localisation approach and thus are near to their optimal solutions. We discuss this localisation procedure next.

4.3.3 IMU Assisted Localisation Using Optimisation

While the 8-point algorithm or the perspective-n-point (PnP) method can be used to determine the camera's pose at a specific frame, these methods suffer from various problems. For instance, the 8-point algorithm cannot be solved for small motions where the baseline is close to zero. Furthermore it is still computationally expensive

as it requires some form of mitigation against outliers, such as RANSAC. The PnP approach on the other hand is faster but is also very sensitive to noise and outliers [47]. In some approaches [51, 28] the PnP method will be more robust to outliers but will prioritise the error of distant features over features closer to the camera. This can lead to inaccurate localisation as distant features carry far greater uncertainty than those in the foreground, due to the nature of projection.

For these reasons we decide to formulate an alternative approach to localisation. Instead of first using a linear algorithm to localise the camera before adding it to a bundle adjustment step, we formulate the problem as a two step bundle adjustment which first optimises the current frame's translation and then optimises the full camera pose. We assume the map points to be correct and do not attempt to optimise their positions (they can be modified by the local bundle adjustment on key frames, as explained in the previous section). In this formulation we are only attempting to localise one camera, namely camera j, and not a set of cameras as in LBA.

In order to insure the translation converges accurately we set the camera attitude to be the absolute orientation estimated by the EKF at the instance the frame was captured, $\mathbf{R}_j = \mathbf{R}_{\text{ekf}}$. We also initialise the translation using the previous key frame's translation.

Using this approach the first stage cost function only has three DoF which significantly reduces the computational cost of the bundle adjustment. We can express the first stage as

$$e_1(\mathbf{t}_j) = \sum_{k \in V_j} \|\mathbf{x}_{j,k} - \text{Proj}(\mathbf{P}(\mathbf{t}_j), X_k)\|^2. \tag{4.3.3}$$

The second stage of the localisation can be seen as optional, as the pose generated in the first stage is likely to be near optimal, with the main error being in the rotational component of the camera's pose. The second stage extends the cost function in equation 4.3.3 by incorporating optimisation over the camera's rotation. To ensure fast convergence we initialise the second stage with the translation obtained from the first stage and the rotation from the EKF. We can then define the second stage bundle adjustment function as

$$e_2(\mathbf{t}_j, \mathbf{R}_j) = \sum_{k \in V_j} \|\mathbf{x}_{j,k} - \tilde{\mathbf{x}}_{j,k}\|^2. \tag{4.3.4}$$

By splitting the frame localisation into two smaller steps we reduce the total number of iterations required for the bundle adjustment to converge, thus reducing the computational expense of this form of localisation. The reduced computational expense is due to our first stage only optimising over three DoF instead of six, and our second stage is initialised with an almost optimal solution. Furthermore, by using the inertial sensor data to initialise the optimisation we remove the need for an initial linear estimation stage. This adds further robustness to the pipeline as the inertial orientation is not affected by poor feature detection caused by motion blur.

4.3.4 M-Estimators

The standard nonlinear least-squares approach used in bundle adjustment is sufficient if we can guarantee there to be few errors in the feature correspondences. This is normally a fair assumption to make as the initial pose estimation step can remove most of the outliers through the use of RANSAC. However, in our formulation we wish to refrain from using RANSAC and thus it is possible for outliers to destabilise the standard least-squares minimisation of the cost functions in equations 4.3.3 and 4.3.4.

A solution here is to use M-estimators in order to increase the robustness of the bundle adjustment to outlying feature matches. M-estimators work by replacing the squared residuals $\sum_i r_i^2$, found in the least-squares approach, by another function ρ of the residuals, yielding

$$\min \sum_i \rho(r_i), \tag{4.3.5}$$

where ρ is a symmetric function with a unique minimum at $r = 0$ and is chosen to increase at a rate slower than r^2. Instead of solving the problem in equation 4.3.5 directly we can reformulate the problem as a weighted least-squares problem using a weighting function w [56].

In order to further increase the robustness of our system we make use of the Cauchy M-estimator. The Cauchy M-estimator is defined by the following equations:

$$\rho(x) = \frac{c^2}{2} \log(1 + (\frac{x}{c})^2), \tag{4.3.6}$$

$$\phi(x) = \frac{x}{1 + (x/c)^2}, \tag{4.3.7}$$

$$w(x) = \frac{1}{1 + (x/c)^2}, \tag{4.3.8}$$

where ϕ is the influence function, w is the weighting function which is applied to each term in a least-squares optimisation and c is a tuning parameter. A visual representation of these functions is provided in figure 4.7.

It can be seen, by comparing the functions in figure 4.7, that the influence of large residuals have a decreasing effect on the error term generated by the Cauchy M-estimator. This in effect reduces the influence of outliers on the optimisation problem thus making it robust against points with large errors.

In this chapter we discussed a factorisation based approach to pose estimation, as well as how techniques such as bundle adjustment can be used in order to increase the accuracy of pose estimates and 3D points by iteratively solving a large scale optimisation problem. We proposed a local bundle adjustment and an IMU assisted localisation procedure to replace GBA and the 8-point algorithm respectively. We further described how M-estimators can be used to increase robustness. In the next chapter we provide details as to how the ideas in the previous chapters can be combined into a complete pose estimation pipeline.

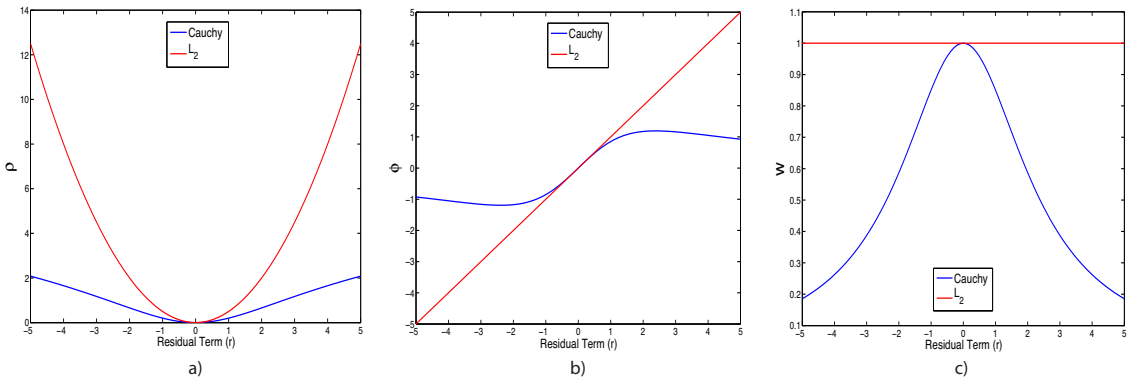


Figure 4.7: A comparison between the (a) Cauchy (blue) and L_2 M-estimator (red), (b) their influence functions, and (c) their weighting functions, which are used to implement them in a standard least-squares optimisation routine.

Chapter 5

System Integration

In this chapter we describe the system design and architecture of our mobile pose estimation pipeline. We discuss how the previously described inertial pose estimation, feature tracking and visual pose estimation systems are combined in order to form the complete system. The motivation for a robust mobile pose estimation pipeline was discussed in section 1.1. Firstly we outline the requirements for our system and then discuss our choice of development platform and how the components described in chapters 2 through 4 are combined to form the completed system. Finally we discuss aspects and considerations related to system initialisation.

5.1 System Requirements

In section 1.2 we outlined the main problem statement as well as the requirements for such a system. Three main requirements exist for a mobile pose estimation pipeline, which is to be used for augmented reality. These requirements are real-time performance, accuracy and robustness.

Real-time performance is defined as the system's ability to run on a mobile device in such a manner as to be able to provide a smooth experience for the user. In this thesis we assume real-time to mean that all the processing is performed with a minimum frame rate of 15 frames per second (fps). This provides us with a maximum processing time of or approximately 67 milliseconds per frame.

Accuracy is defined in two parts: firstly as the ability to track feature points between consecutive frames with few outliers, and secondly as the ability to obtain an accurate estimate of the camera pose. If the system cannot track a sufficient number of points with a high enough accuracy it will lead to drift, whereby the map and augmentations appear to move independently, thus reducing the quality of user experience.

Robustness is defined as being able to accurately track features and predict the camera pose even in the presence of noise, large motions, and changes in illumination. Meeting all these requirements is incredibly difficult and requires very high quality sensors, tracking and recovery algorithms. For these reasons we limit our definition

of robustness to cases under normal usage. As the camera is moved by a person, who can get visual feedback via the screen as to motion blur and tracking quality, we make the assumption that the user will generally keep motion within the constraints of the system. Thus the system only needs to be robust against accidental large motions over a short period of time, moderate noise and gradual changes in illumination.

As the computational capabilities of mobile devices are advancing rapidly, we place a greater importance on the accuracy and robustness of the system and a secondary importance on the speed at which the pipeline operates. Still, we do make every effort to keep the computational complexity of the system as low as possible.

5.2 System Model

Our system pipeline consists of four main parts, each running in its own thread, and combined to form the total system pipeline shown in figure 5.1. The four main parts of the pipeline are as follows.

Video acquisition and processing (30Hz): The camera notifies the process that a new video frame is available which triggers the start of this thread. This task uses the GPU to convert the video frame into a greyscale image and to perform Harris corner detection on the frame, as discussed in section 3.1.1.2. Once the corner points have been detected the video frame and the detected feature locations are stored in a frame object which is then added to the *frame queue*.

Inertial sensor processing (60Hz): The inertial pipeline runs at two times the speed of the video acquisition pipeline. This is to ensure that the inertial pipeline is robust to fast motions which will cause the video pipeline to fail. The inertial pipeline captures data from all the sensors on the iPhone and processes this data using the EKF described in section 2.3. The output states (attitude quaternion and angular velocity) of the Kalman filter are then sub-sampled at the same rate as the video acquisition pipeline. This ensures that each frame has corresponding inertial measurements.

Frame processing: This stage of the pipeline is where the inertial and visual data are integrated. Frame objects are pulled from the *frame queue* and processed. The first stage of this processing involves using the angular velocity measurement, the feature points and the pre-existing tracking set to determine the relationship between the features detected in the current frame and the features in the map, as in our hybrid tracking algorithm from section 3.3. These feature correspondences, along with the map and the inertially estimated pose are then fed into our localisation algorithm described in section 3.3. The frame object is appended with the final pose estimate and the feature correspondences, and is placed in the key frame queue.

Key frame and map management: This stage of the pipeline is only run when the frame popped off of the key frame queue is deemed to be a key frame. If the frame meets the criteria of a key frame then this stage uses the current frame and pose estimate, as well as the previous key frame, to extract common features between

the two which do not occur in the map. These features are then triangulated and added to the map. Next we perform local bundle adjustment in order to optimise the updated map and the key frame pose estimates. The final step is to add these features to the current tracking set so that they can be used for localisation of future frames.

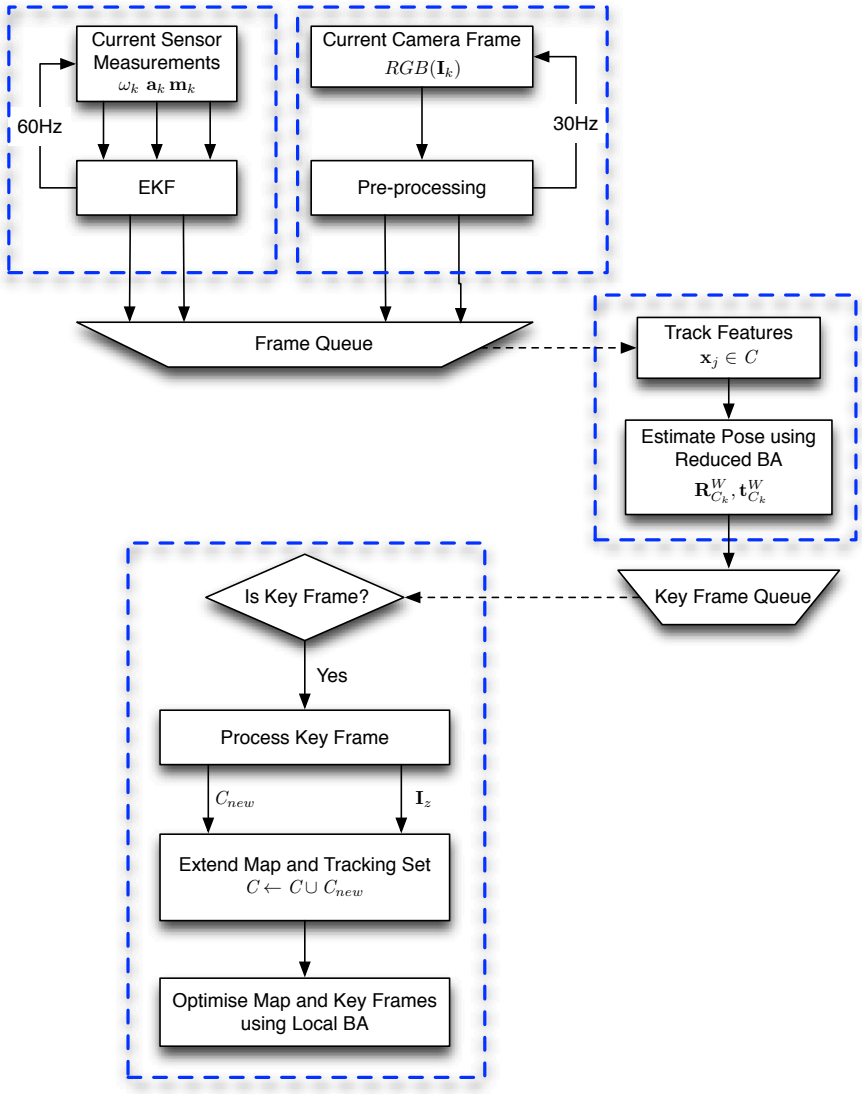


Figure 5.1: Our pose estimation pipeline. The arrows dictate the direction of information flow, dashed lines represent data being fetched from a queue and the blue boxes indicate the threads for the various tasks.

Our pipeline is similar in structure to that of Klein and Murray's PTAM system [19], with the biggest difference being the additional thread for inertial pose estimation. As each of our stages executes in its own thread, the frame rate localisation

can continue without having to worry about being blocked by the computationally expensive final stage. The final stage is given the lowest priority and thus only executes when the processor has time between incoming frames. This provides a smooth experience as the camera can continue to localise itself while the map is being extended. The fact that the iPhone 5 has a dual core processor further reduces the chance of lag due to long running, computationally intensive tasks.

5.3 Pre-processing

The biggest trade-off when choosing image quality is between processing speed and system accuracy. We explored the three pre-set quality options available on the iPhone (being low, medium and high quality) and found that the medium quality (640×480 pixels) image provided us with sufficient accuracy while still allowing the initial frame processing to be done quickly.

The initial processing consists of two stages, the first being to convert the full colour frame to greyscale, and the second being the detection of feature points. The greyscale conversion can easily be done on the GPU with almost no overhead as the GPU and CPU share the same memory space in most mobile devices.

For the feature detection two approaches were considered, as discussed in section 3.1.1. It was found that if Harris corners are detected using the Shi-Tomasi corner measure and the algorithm found in the GPUImage library, the processing time is about 4ms per frame, with a maximum corner count of 512 corners. While the FAST corner detector exhibits slightly better performance, with an average processing time of around 2ms and no maximum limit on the number of features, the algorithm does put significant load on the CPU.

Thus we decided to use the Harris corner detector with the Shi-Tomasi corner measure in our pipeline. In total the pre-processing only accounts for about 5% of our frame budget, and around 7% of a single core's usage. The CPU usage is primarily due to the copying of feature point locations from the GPU to the CPU. Although the memory is shared there is still an overhead in reading pixel values from the GPU frame buffers.

5.4 Map Management

Once the system is tracking and selecting key frames, as described in section 3.3, we use the key frames to extend the sparse tracking map. The process involved in extending the tracking map is similar to the initialisation procedure we describe in section 5.6.2, except that we do not need to use the 8-point algorithm to calculate the pose. Since a key frame is selected when we still have a fairly large overlap with the previous key frame, there should still be common points between the two frames and thus the pose is known from the frame processing step in our pipeline. We use

BRIEF descriptors and the frame pose estimates to extract feature matches between the current key frame and the previous key frame which are not in the current map.

These new matches are then triangulated, using our pose estimates for the frames, and added to the map. A further extension to this approach would be to find additional features using the knowledge of the pose and an epipolar search, when an insufficient number of features are found using normal feature detection methods. This was not implemented due to time constraints, but can be considered as future work for the project.

Once the new key frame has been processed and the map has been extended we run the local bundle adjustment procedure outlined in section 4.3.2. The outcome of this bundle adjustment is an optimised map as well as optimised key frame poses for the two newest key frames in the bundle adjustment window. These optimised poses and map points provide good quality reference data for tracking and possible offline dense reconstruction.

5.5 Re-localisation

Should a suitable key frame not be found before tracking is lost, the system will fail as the link between the current map and incoming frames is lost. For this reason a re-localisation algorithm is also required. This algorithm is used to find the most likely position of the camera relative to the existing map such that tracking can continue. Klein and Murray [19] use a correlation based approach to determine the closest key frame to the current frame after tracking is lost. In order to make this more efficient the correlation is computed between a tiny down-sampled version of their key frames and the down-sampled version of the current frame. Once the closest key frame is found, its pose is used to re-project the map into the current frame and determine correspondences, and hence re-initialise tracking.

Due to time constraints we did not implement a re-localisation algorithm, but rather relied on the user's ability to constrain motion and optimise the environment in order to prevent tracking from being lost. A more robust re-localisation solution is laid out as future work for this thesis. Although it was not implemented we do consider that Klein and Murray's implementation works sufficiently well, and can be adapted to use the pose estimate obtained from the inertial sensors in order to reduce the search space for viable key frames. Furthermore, as the pose is already known we can rotate the video frames and key frames such that they are all oriented relative to the same reference orientation. This will likely improve the performance and robustness of the correlation matching and provide a better estimate of the most likely key frame. Furthermore, the inertial pose and translation retrieved from the key frame can then be used to extract matching points which are in the current map, thereby re-initialising tracking.

5.6 System Initialisation

Before the system can estimate the poses of incoming frames we need to initialise the various components. This section provides an explanation of how the various components are initialised, from the point at which the user starts the tracking.

5.6.1 Inertial Initialisation

The first step in initialising the inertial pose estimation system is defining the world coordinate system. As discussed in section 2.1.1, this system is easily defined by setting the world reference vectors to the values obtained from the first inertial measurement, when the user starts the tracking process, such that

$$\mathbf{a}_W^0 = \mathbf{a}_B^0, \tag{5.6.1}$$

$$\mathbf{m}_W^0 = \mathbf{m}_B^0. \tag{5.6.2}$$

Using this model for the world coordinate system ensures that the initial orientation is always the identity, and all motion is relative to the first frame. This provides an easier way to ensure the camera and inertial pose estimates have the same origin. The next step is to initialise the EKF as we discussed in section 2.3.3.

5.6.2 Visual and Map Initialisation

The initialisation of the visual sub-pipeline and the system map are significantly more involved than the inertial initialisation procedure. There are two main parts involved in the initialisation of the rest of the system: firstly we need to initialise two key frames and a tracking set, and secondly we need to generate the initial sparse map which is used to assist tracking and localisation.

The first stage requires the user to hold the phone still and tap the screen to begin the tracking process. The first frame captured is stored as the first key frame in our system, and is initialised such that $\mathbf{P} = [\mathbf{I}_3|\mathbf{0}]$ is the origin. The next step requires the user to move the phone with a significant translational motion such that feature scale change is kept to a minimum. During this step rotation is acceptable, as long as the main pose change is due to translation. Once a significant baseline has been produced the user taps the screen again to capture the second key frame. Using these two key frames and the features extracted in the pre-processing stage we generate a set of corresponding features using the BRIEF feature descriptor and Hamming distance matching as discussed in sections 3.1.2.1 and 3.1.3 respectively.

These matches are then used along with the 8-point algorithm and RANSAC as described in section 4.2 in order to determine the essential matrix, and thus the second key frame's pose relative to the first. Using this pose we triangulate the correspondences to form the initial map. The relationship between the detected feature points and their corresponding map points is stored so that it can be used

for tracking and localising features in future frames. The final step involves running bundle adjustment on the two key frames and the map points to optimise their poses and locations before we being tracking, as was discussed in section 4.3.2. We also initialise the tracking set with the features found in the second key frame.

From this point on the system has been initialised, and tracking and localisation can continue as described by the system pipeline in section 5.2. The full system initialisation algorithm is described in pseudo code in algorithm 2.

Algorithm 2 System Initialisation

1: User selects initial key frame
2: EKF is initialised with state $\mathbf{x} \leftarrow \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}^T$
3: World gravity reference set to $\mathbf{a}_W^0 \leftarrow \mathbf{a}_B^0$
4: World magnetometer reference set to $\mathbf{m}_W^0 \leftarrow \mathbf{m}_B^0$
5: First Key frame's pose is set as the origin $\mathbf{P}_0 \leftarrow [\mathbf{I}_3|\mathbf{0}]$
6: User moves camera in a mainly translational manner
7: User selects second key frame
8: A set of feature correspondences A is created
9: The essential matrix $\mathbf{E}$ is found using RANSAC and the 8-point algorithm
10: Extract the pose $\mathbf{P}_1$ of the second key frame from the essential matrix
11: Triangulate features and add their 3D locations to the map
12: Initialise the tracking set with the features detected in the second key frame

Chapter 6

Results

This chapter focuses on the testing and evaluation of the sub-systems proposed in the previous chapters, as well as qualitative testing of the completed pose estimation pipeline performance. The order of testing follows the same order as the thesis layout.

The chapter starts with a description of the datasets we captured and a description of the various quantitative evaluation measures we use throughout the testing procedure. We then present the results of our tests, and finally present an overall discussion of our results.

6.1 Experimental Setup

This section describes various aspects relating to the testing of our algorithms and pose estimation system. We start by describing the testing setup and proceed with a description of the datasets we captured and an explanation of the quantitative performance metrics we use throughout testing.

6.1.1 Testing Platform

We make use of an iPhone 5 as well as a desktop computer in order to test our algorithms. As all the algorithms are implemented in C++, using well supported libraries, they can be run on both the mobile platform as well as on the computer without having to modify the core algorithm code.

We make use of the mobile device for testing the computational and run-time performance of our algorithms, and the computer along with pre-captured datasets to test the robustness and accuracy of our system.

Along with these test benches we make use of both qualitative and quantitative evaluation to draw conclusions about the effect of including inertial data into the pose estimation pipeline.

In order to ensure we have repeatable and reliable data for testing on we use data captured from the iPhone's camera and inertial sensors. In order to capture these

datasets we developed a tool called CameraSense, which simultaneously captures video data at 30Hz and inertial data at 60Hz. We capture both the raw and iPhone filtered data, so that we can make comparisons between our algorithm's output and the data the device returns. The CameraSense tool has been released under an Open Source license in order to facilitate further research into the development of mobile pose estimation algorithms. The source code can be found on GitHub at `https://github.com/system123/CameraSense`.

Ground truth data for our datasets was captured using a Vicon motion capture system, which accurately records the absolute position of the phone relative to the scene. This data can then be used to generate ground truth data for the pose of the phone relative to any location in the scene, or in our case relative to the first frame of the video sequence. The Vicon system captures data at 120Hz and this data was manually synchronised to the inertial sensor data. These datasets can be found on GitHub at `https://github.com/system123/ViconDatasets`.

Using the CameraSense tool and the Vicon system we captured various datasets. These datasets cover a wide range of motion and different scene types, from planar checker-board scenes to cluttered desk scenes. From the various datasets captured we selected three for the purposes of this chapter as they cover a wide range of situations which our system might encounter. These datasets are defined and described as follows:

- dataset 1 consists of fast rotational motion in the presence of a checker-board pattern;
- dataset 2 is a normal use case consisting of controlled motion around a single object in a small area;
- dataset 3 consists of a significant translational motion across a typical cluttered scene.

We also made use of the bust dataset of Lhuillier and Quan [25] in order to test our bundle adjustment algorithms. This dataset consists of a series of 26 photographs taken in a loop around a bust. The photographs are spaced such that the first and last image are at almost the same location. Due to the nature of the dataset it provides a good qualitative measure for the accuracy of bundle adjustment. A few frames from each of our visual datasets can be seen in figure 6.1.

In order to evaluate the performance of our inertial system with respect to other sensor based approaches we make use of ground truth data obtained from a high accuracy, commercial IMU manufactured by YEI technologies, known as the 3-Space Sensor (TSS)[1]. This sensor allows us to evaluate the quality of the inertial sensors on the phone as well as compare our filter's accuracy to that of a commercial pose estimation filter. The TSS implements a full quaternion based extended Kalman

[1] `http://www.yeitechnology.com/yei-3-space-sensor`

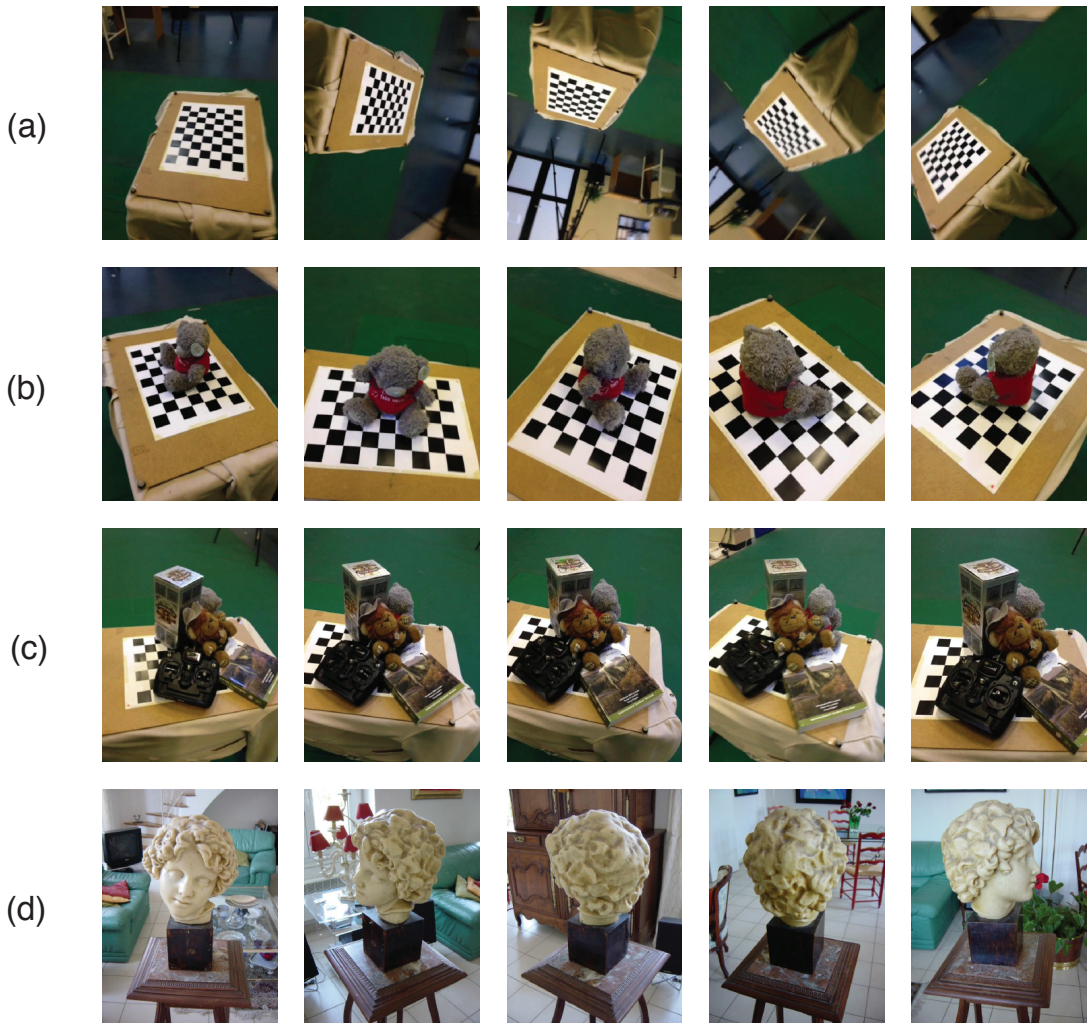


Figure 6.1: A few images from each of our visual datasets. The contact sheets correspond as follows: (a) dataset 1, (b) dataset 2, (c) dataset 3 and (d) the bust dataset.

filter which estimates both attitude as well as sensor bias. The TSS has an average accuracy of $\pm 1^\circ$ when fully calibrated.

In order to use the TSS to measure the same motion as the device we designed and 3D printed an iPhone case which allows the sensor to be attached to the back of the mobile device during the testing sequence. This case is shown in figure 6.2. Furthermore the TSS software provided by YEI-technologies allows us to set the sensor's axes to be in the same orientation as the senors on the mobile phone. Before each dataset was captured we re-calibrated the TSS magnetometer as well as the magnetometer on the mobile phone, in order to minimise their effects on each other.

As the body of existing work exploring the use of inertial sensors and mobile

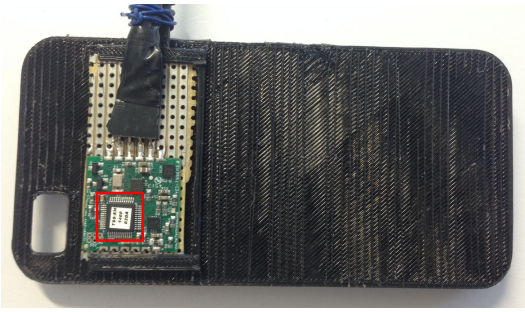


Figure 6.2: The TSS is attached to the iPhone with the use of a specially designed and 3D printed iPhone case. The IMU sensor (red box) is aligned with the camera to ensure minimal offset from the actual position of the iPhone's inertial sensors.

devices for pose estimation is limited, we make comparisons between our algorithms which include inertial sensor data, and the same algorithms without the inclusion of the sensor data. Furthermore, we draw comparisons between state of the art methods implemented on a computer and our algorithms running on the computer, when both are supplied with our datasets. It is often the case that these state-of-the-art methods are too computationally expensive to run on an iPhone 5 generation of mobile device. Thus the results are compared on a computer and extrapolated to the expected performance on a mobile platform.

6.1.2 Performance Metrics

We make use of various performance metrics in order to quantitatively evaluate our system's performance with respect to our ground truth data. These metrics can be broken down into the following three categories:

- pose evaluation metrics;
- feature tracking metrics;
- computational performance metrics.

Pose evaluation metrics: In order to evaluate how closely our estimated pose agrees to the ground truth pose, we make use of two different metrics. The first is the quaternion similarity (equation 6.1.1), and the second is the mean squared error (MSE). The dot product of two quaternions represents the angle between two points on the quaternion hypersphere, which double covers the space of all 3-dimensional rotations. Using the fact that we know two quaternions represent the same rotation if $\hat{\mathbf{q}}_1 = \hat{\mathbf{q}}_2$ or if $\hat{\mathbf{q}}_1 = -\hat{\mathbf{q}}_2$, we can define the quaternion similarity metric as

$$e_q = \arccos\left(\frac{|\mathbf{q}_1 \cdot \mathbf{q}_2|}{\|\mathbf{q}_1\|\|\mathbf{q}_2\|}\right). \tag{6.1.1}$$

This metric provides us with the angle between the two quaternions we are comparing, taking into account the criteria for quaternion similarity. Apart from this measure, we also make use of the MSE error between roll, pitch and yaw angles described by the Euler angle representation of each of the quaternions. While this measure appears intuitive we need to take care in drawing conclusions from it due to the fact that Euler angles do not provide a unique representation of a single rotation and also suffer from gimbal lock as described in section 2.1.2. To evaluate the difference in translation we make use of a standard MSE measure between our ground truth translation $\mathbf{t}$, from the Vicon system, and the translation $\hat{\mathbf{t}}$ computed by our localisation algorithm. The MSE of the translational components can be calculated as

$$e_t = \frac{1}{3}\sum_{j=1}^{3}(\hat{\mathbf{t}}_j - \mathbf{t}_j)^2. \tag{6.1.2}$$

Feature tracking metrics: Quantitative evaluation of feature tracking algorithms is challenging due to the large number of different tracking techniques available, as well as the fact that it is hard to obtain reliable ground truth data from real life datasets. For this reason we employ both quantitative and qualitative techniques to evaluate our feature tracking. Our quantitative evaluation relies on the speed at which features are tracked in a video sequence, as well as the rate at which features are lost during tracking. We compare these results to other tracking methods applied to the same sequence of video and initialised with the same initial tracking points.

Computational performance metrics: We use standard performance measures such as memory usage, memory growth, computation time and CPU usage in order to evaluate the computational performance of our system. These quantitative metrics will provide us with the information required in order to draw conclusions as to how well our proposed system performs on a mobile device.

6.2 Inertial Pose Estimation

In this section we compare the various components of our inertial sub-system with respect to two ground truth models. The first set of tests compares the accuracy of the iPhone sensors and our pose estimation model to that of a high quality IMU. The seconds set of tests compare our pose estimates to the ground truth data obtained from the Vicon motion capture system.

6.2.1 Magnetometer Pre-filter Test

During our design phase we noticed that the magnetometer readings provided by the iPhone 5 contain discontinuities and inaccuracies in the presence of disturbances. In order to correct the magnetometer readings we designed a pre-filter to remove

the discontinuities, to ensure a random-walk type motion, as discussed in section 2.2.4.2.

The test described in this section focuses on evaluating the improvement in accuracy gained when using the magnetometer pre-filter with our EKF implementation. The test is performed using the TSS in a strapped down configuration on the mobile device, as described in section 6.1.1. The calibrated magnetometer values provided by the TSS are compared to the values obtained from the iPhone 5 with and without our magnetometer pre-filter. We perform the test for fast and slow motion, and evaluate the output using the MSE. The results can be seen in figures 6.3 and 6.4.

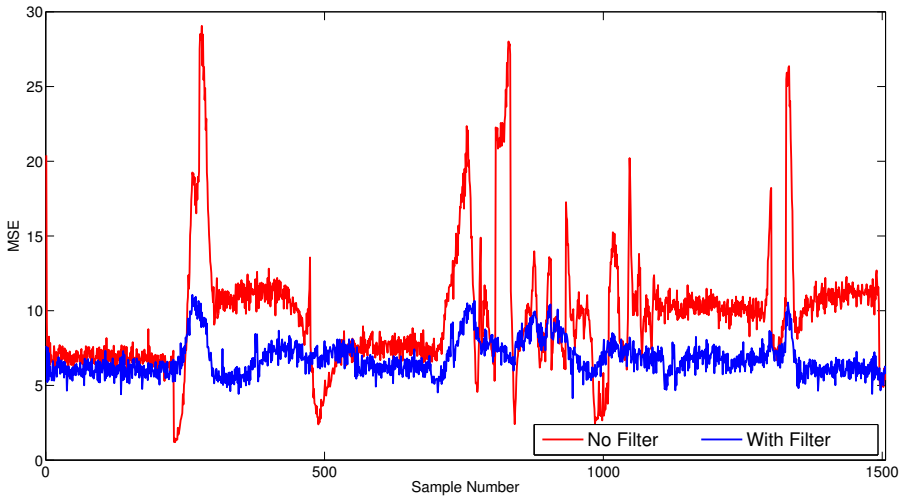


Figure 6.3: The mean squared error, measured against the TSS magnetometer, obtained from using the filtered and unfiltered versions of the iPhone's magnetometer in our EKF for slow, controlled motion.

It can be seen in figure 6.3 that the magnetometer filter leads to improved readings during slow and controlled motion. It was observed that the discontinuities in the magnetometer readings during slow motion were mostly due to soft iron disturbances and thus could be corrected for.

In the case of fast motion (figure 6.4) the pre-filter breaks down and the corrections to the magnetometer readings cause an accumulative error to occur. Upon further observation it was determined that the discontinuities present in the device's magnetometer readings also occur because of fast motion. These discontinuities are indistinguishable from those caused by disturbances and thus cause the filter to erroneously correct for non-existent disturbances.

For this reason it was decided that the magnetometer readings from the TSS will be used in place of those from the iPhone's magnetometer for future tests. This will remove the effect of the iPhone magnetometer and allow us to test our algorithms using more reliable sensor readings. In the case of our Vicon tests we

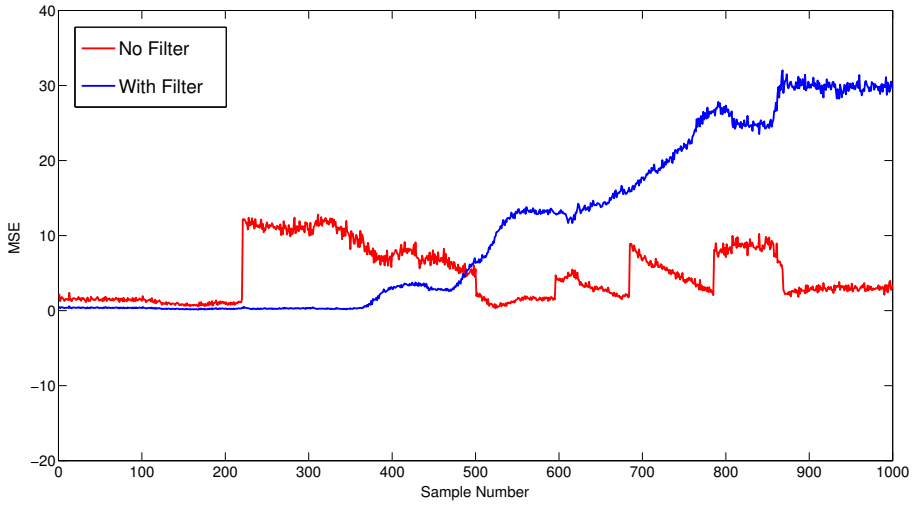


Figure 6.4: The mean squared error, measured against the TSS magnetometer, obtained from using the filtered and unfiltered versions of the iPhone's magnetometer in our EKF for fast motion.

do not have accurate magnetometer readings and thus synthesise this data from readings provided by the Vicon system, by adding Gaussian white noise in order to simulate the magnetometer sensor noise[2].

6.2.2 Pose Estimation Complexity

Since our EKF is running on a mobile device it is important to ensure that the computational cost of running the algorithm is low enough to ensure we have sufficient processing power remaining for our vision algorithms. In order to test for this we ran the EKF on the iPhone at three different sampling rates, 30Hz, 60Hz and 100Hz, while profiling the code. We profiled CPU usage and memory usage at the three sampling rates for a period of 2 minutes each. The results of these runs can be seen in table 6.1.

From table 6.1 we can see that one iteration of our EKF algorithm always runs in less than 1ms. This is expected as the complexity of the algorithm is independent of the sampling rate. Furthermore the small state vector and the ARM NEON optimisations offered by the Eigen linear algebra library we utilise lead to fast matrix operations, and thus keep the computational cost of the EKF very low. The memory usage of the application remains constant at about 3MB, which is as expected as the EKF's memory usage is independent of the sampling rate. The increased CPU

[2]The issues relating to the iPhone 5's magnetometer were fixed in a later version of iOS which unfortunately came out too late for use in this particular study. This means that our use of a more reliable sensor is likely to resemble what can be expected from the sensor on the device after the iOS update.

Sampling Rate	CPU Usage	Memory Usage	Running Time
30Hz	9.7%	2.30MB	<1ms
60Hz	13.2%	2.28MB	<1ms
100Hz	17.3%	2.35MB	<1ms

Table 6.1: Comparison of the computational complexity of our EKF implementation for different sampling rates as executed on the iPhone.

usage between the various sampling rates is found to increase linearly with respect to sampling time, as expected.

Ultimately, a sampling rate of 60Hz was chosen as it provides a good balance between CPU usage and sampling rate. The choice of sampling rate is important as it is one of the limiting factors in the maximum dynamics that the system can handle.

6.2.3 Pose Estimation Accuracy

The objective of the next test is to measure the accuracy of attitude estimation from our EKF implementation. We further test the accuracy of attitude provided by the CoreMotion library on the iPhone 5. To do this we make use of the datasets we captured with the Vicon system. We placed markers on the phone and in the environment, and from that we could compute the attitude of the phone with respect to the world using the TRIAD method described in section 2.3.2.4. The results from this test conducted on dataset 2 can be seen in figures 6.5 and 6.6. It should be noted that the magnetometer measurement was replaced by a measurement synthesised from the Vicon data.

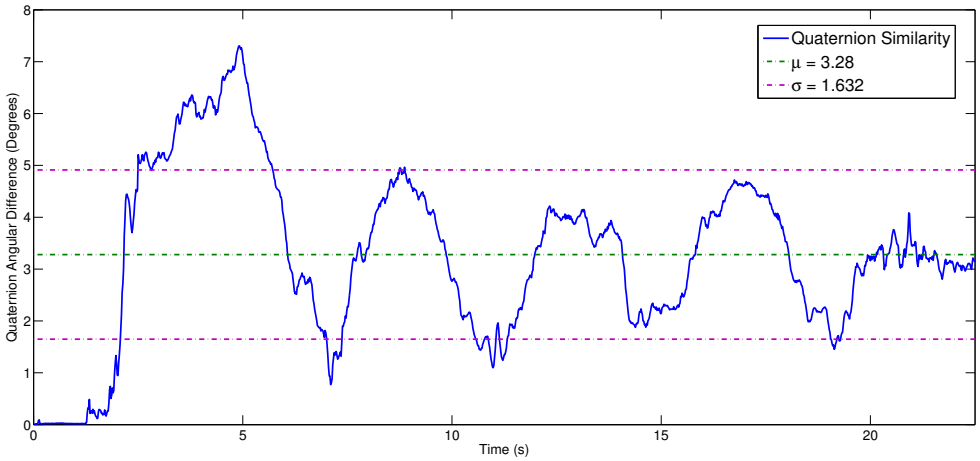


Figure 6.5: The quaternion angular difference (in degrees) between the attitude determined from the Vicon measurements and our estimation from the inertial sensors.

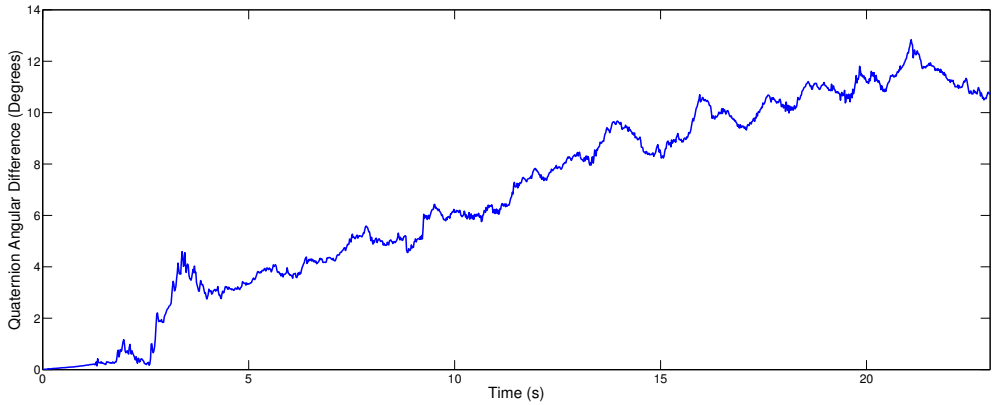


Figure 6.6: The quaternion angular difference (in degrees) between the attitude determined from the Vicon measurements and the attitude from the CoreMotion library.

As we used synthesised magnetometer readings to obtain the results in figure 6.5 we repeated the test using the TSS as our ground truth reference and a moderate motion. The TSS was attached to the phone as described in previous tests. With this configuration we substituted the phone's magnetometer readings with those obtained from the TSS. The results of this test are depicted in figure 6.7 using the same quaternion error metric.

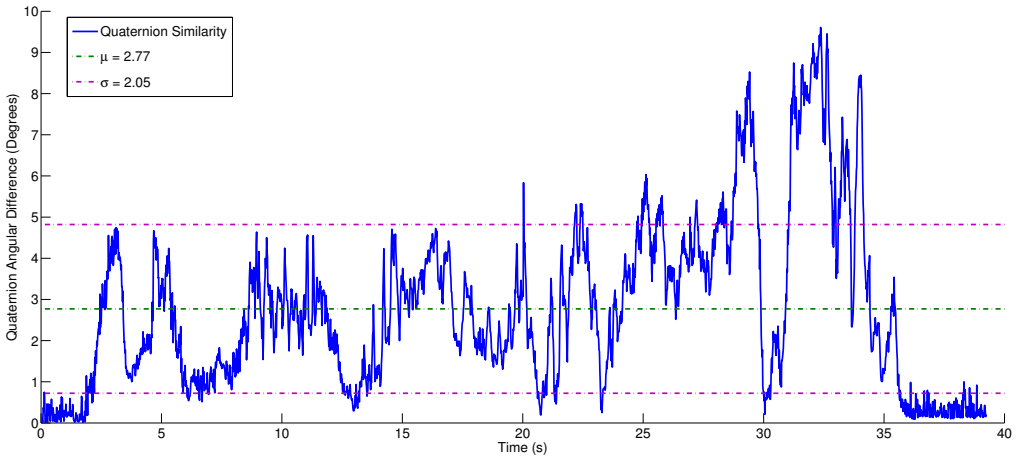


Figure 6.7: The quaternion angular difference (in degrees) between the TSS and our EKF estimates of attitude. The larger error between 30 and 35 seconds is due to fast motion. Note however that the filter is able to recover after that.

By comparing figures 6.5 and 6.6 we can see that the attitude provided by the CoreMotion library exhibits significant drift, and has poor accuracy when compared to our estimates. The reason for this can be attributed to the faulty magnetometer calibration.

Our EKF implementation shows an average dynamic accuracy of $\pm 5^\circ$ in situations where the motion is still acceptable for vision based processing. It is important to note that our attitude estimate does not exhibit large drift or diverge even when under the influence of large dynamics, as can be seen in figure 6.7. This behaviour is confirmed by noticing that the estimation error approaches zero when the device returns to a resting position around the 36th second. Furthermore, it can be seen in figures 6.5 and 6.7 that the estimation error exhibits a periodic type behaviour where the error drops to around 2° every few seconds. This is due to the device motion slowing or coming to a stop briefly as we move it around in the environment.

The reason our estimate error returns to zero even after large motions is due to the fact that we place more certainty on our measured states than our process model estimates. When the motion is small the measurement will be near to the actual attitude and thus the influence of the process model will eventually be negligible. The static accuracy of our EKF was determined to be less than $\pm 2^\circ$ on average during periods of slow or no motion.

We find our results to be similar in nature to those obtained by Von Marcard [54] with his implementation of a quaternion based EKF.

6.3 Feature Tracking

In this section we test our proposed hybrid tracking system in terms of accuracy, robustness and computational intensity. Comparing feature tracking algorithms quantitatively is challenging due to the large number of trade-offs between various algorithms. These trade-offs are usually between accuracy, speed, number of features we are tracking and the type of features which get tracked. For this reason we use our feature tracking approach without the inclusion of inertial data as a baseline measure and then compare its performance to the same algorithm with inertial data included.

Furthermore we also compare the accuracy and computational efficiency of our algorithm to that of the KLT tracker in both a qualitative and quantitative manner. Previous studies have shown KLT to be superior in terms of accuracy and robustness, but its computational complexity prevents it from running in real-time on a mobile device.

6.3.1 Robustness

The objective of this test is to evaluate the effect of including inertial information into our feature tracking algorithm. In order to evaluate this we make use of our Vicon datasets and compare the number of accurately tracked features between a version of our algorithm without inertial input and the full algorithm. It should be noted that the implementation which excludes the inertial sensors initialises the search space in the current frame using the location of the feature in the previous frame. This mimics the initialisation which the KLT tracker uses for feature tracking.

The feature trackers were initialised with the same manually selected key frame and the detected features in this frame formed the initial tracking set. These feature points were then tracked for 120 frames and the number of accurately tracked features per frame was recorded. Features were determined to be inaccurate or lost once the NCC between the tracking patch and the template fell below 0.7 and were removed from the tracking set if they were lost for more than ten consecutive frames.

We performed this test using dataset 1 to evaluate the performance when exposed to fast rotation, dataset 2 to evaluate performance under general operating conditions which include rotation and translation, and dataset 3 for pure translation. The results for the first two datasets are presented in figures 6.8 and 6.9. There was no discernible difference in the tracking length from the two versions of the algorithm in the case of dataset 3, and thus the results have been omitted. This is to be expected as the gyroscope reading during a purely translational motion is negligible and thus the two algorithms should be equivalent.

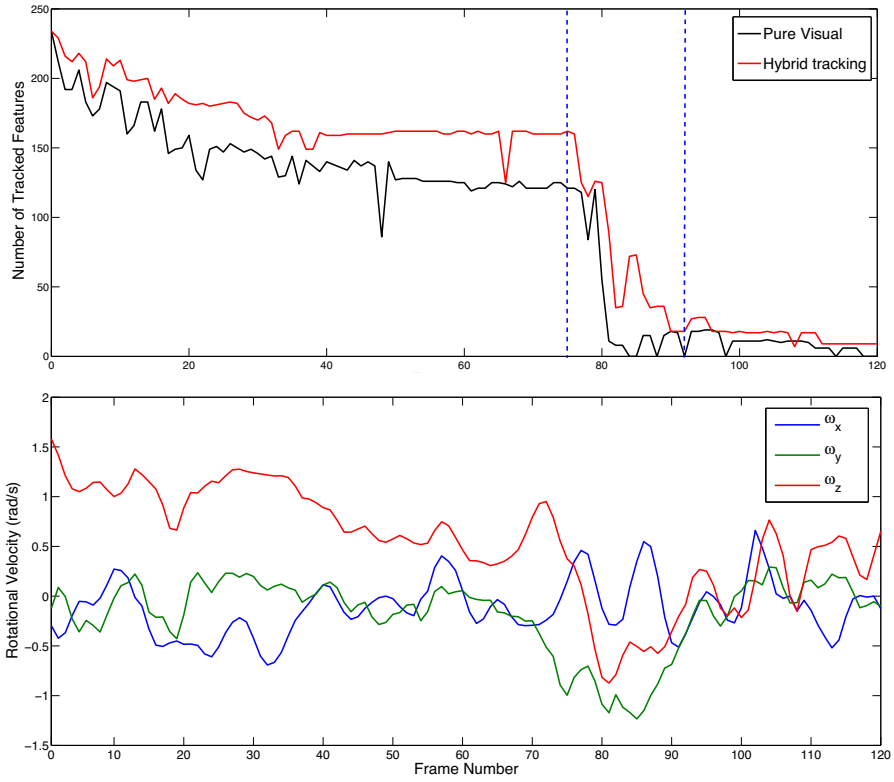


Figure 6.8: Results from our hybrid tracking algorithm with and without inertial sensor information for dataset 1. The top graph shows the number of features tracked over time and the bottom graph shows the corresponding gyroscope readings in radians per second. The dashed blue lines highlight a period of large rotation where most of the tracked features were lost.

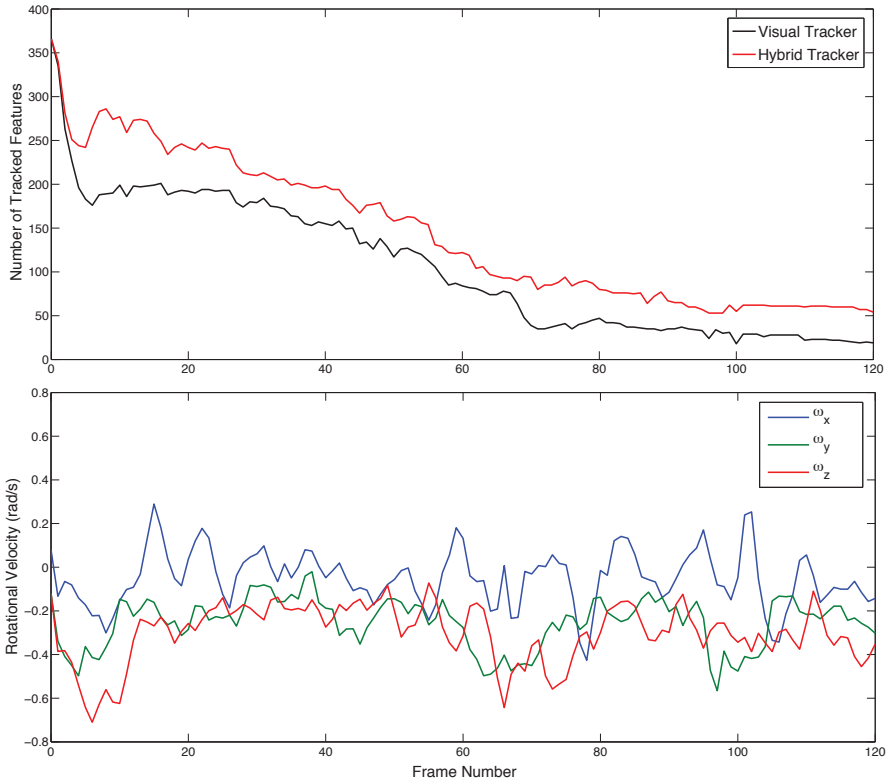


Figure 6.9: Results from our hybrid tracking algorithm with and without inertial sensor information for dataset 2. The top graph shows the number of features tracked over time and the bottom graph shows the corresponding gyroscope readings in radians per second.

In figure 6.8 the frames between the two dashed lines coincide with a large and fast rotational motion as well as translational motion due to the camera being rotated by hand. This lead to most of the tracked features being lost except for a few distant features which were largely unaffected by the translational component of the motion.

We can see from figures 6.8 and 6.9 that the inertial assisted tracker almost always performs better than the purely visual implementation of our tracking algorithm. Less feature points are lost due to large rotations when using the full hybrid algorithm than when using the vision only implementation.

We find these results are similar to those reported by Hwangbo et al. [18].

6.3.2 Accuracy

Measuring the accuracy of a feature tracking algorithm is a challenging problem as no true ground truth data exists for the location of feature points in an image. While we can compare feature locations between various algorithms this does not provide certainty as to which algorithm's location is actually the correct one. For this reason we qualify accuracy by observing the movement of features between

video frames. While this approach cannot provide a quantitative measure of feature tracking accuracy it does provide an overall impression as to the accuracy and drift of the algorithm.

We perform this test by plotting the inter-frame motion of the tracked feature points for each frame of the tracking sequence. By analysing these frames we can clearly see outliers and detect which features have drifted. These plots were produced for both our hybrid tracking algorithm as well as for the KLT tracker.

Figure 6.10 shows a few frames comparing features tracked by our hybrid tracking algorithm to those tracked using the KLT tracker.

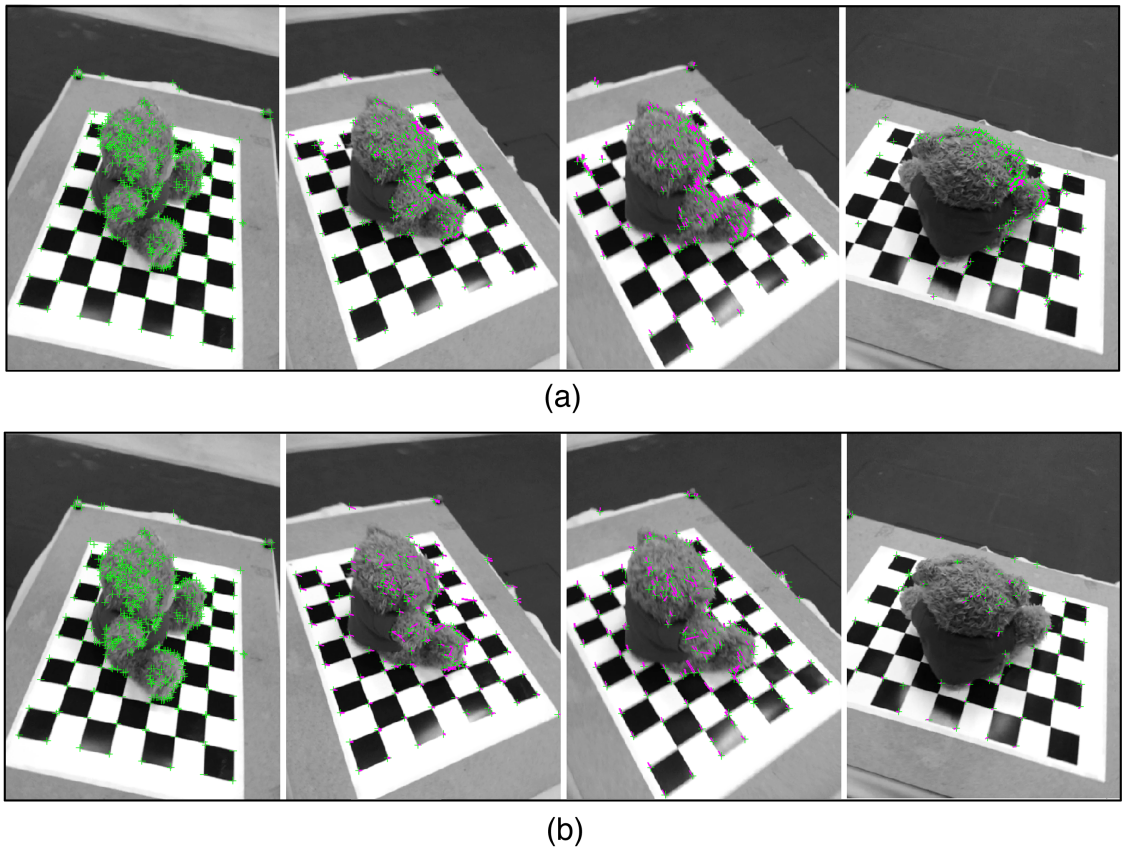


Figure 6.10: A comparison between the KLT tracker (a) and our hybrid feature tracker (b) when initialised with the same tracking set. Tracking was evaluated at increments of 30 frames, and thus the tracking sets above represent a 120 frame tracking sequence. Significant drift can be seen in the KLT tracker's final frame, while our algorithm has fewer features but of a higher accuracy.

Upon analysing these frames it was found that while our tracker had significantly less features remaining in the tracking set at the end of the test sequence, the remaining features were accurately tracked with minimal drift. On the contrary the KLT tracker had more features remaining but there was significant drift in their

locations. This can be seen clearly in the last frame where a large portion of the KLT features are clustered together around the edge of the bear. It can also be seen by comparing the features on the checker-board in the KLT implementation to our hybrid tracker implementation.

6.3.3 Computational Efficiency

Although the main aim of our tracking algorithm is to increase robustness to fast motion, we still require it to be fast enough to track features at frame rate on a mobile device. Furthermore, we also require that the algorithm has a low computational complexity such that there are still computational resources available for the rest of the system.

In order to test the speed of the algorithm we timed the execution of the tracker over three short sequences of 30 frames and we initialised the tracking set with 60 features. This was done in order to reduce the effect of feature loss on the timing estimates. Results are then compared to the implementation without the sensors as well as to a KLT implementation running on the iPhone. The KLT tracker was implemented using the OpenCV library [6] and was initialised with the same initial tracking set as our tracker.

The results of our two hybrid tracking implementations and the KLT tracker are summarised in figure 6.11. It is clear that our method has a significantly faster running time than the KLT tracker. Furthermore we can note that the addition of inertial information into the tracker has a negligible effect on the running time of our algorithm.

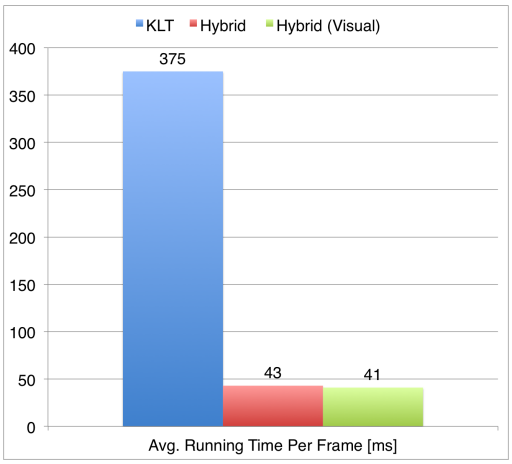


Figure 6.11: A comparison between the average running time of our hybrid tracking algorithm versus the KLT tracker for tracking a set of 60 features.

6.4 Visual Pose

In this section we use the ground truth data from the Vicon system in order to compare the accuracy of our local bundle adjustment (LBA) algorithm to that of the global bundle adjustment (GBA) algorithm. We also compare the computational efficiency of these algorithms.

We test our optimisation based approach to camera localisation in terms of accuracy, robustness and computational efficiency. We compare our algorithm which makes use of the Cauchy M-estimators and inertial sensor initialisation against the pure approach.

6.4.1 LBA Accuracy

The objective of this test is to measure the accuracy of our LBA algorithm. As this algorithm only runs on key frames in our system we manually select key frames from dataset 2. We also make use of the bust dataset and assume the various images are all key frames.

In order to perform the test we require prior 3D positions of points as well as their corresponding projections in the various key frames. We obtain these priors by extracting and matching SIFT features between the key frames and then using the 8-point algorithm and RANSAC in order to generate initial pose estimates for the cameras. This approach is taken in order to minimise inaccuracies from other aspects of the system, and thus to test only the optimisation accuracy.

The 3D positions, feature correspondences and camera pose estimates are then used to initialise the LBA and GBA algorithms. The final results from the bust dataset are compared visually as no true ground truth exists. The results from dataset 2 are compared to the camera poses obtained from the Vicon system. The results of these tests are provided in figures 6.12 and 6.13.

It can be seen from figure 6.12 that the final camera poses obtained from the LBA algorithm are near to those which GBA returns. It can be seen that our algorithm optimises the camera poses to positions which are significantly closer to those obtained by global bundle adjustment when compared to the priors. When doing this comparison it is useful to note the position of the first and last camera relative to each other, as these cameras are located rather close together.

From figure 6.13 it can be seen that the LBA algorithm provides a good middle ground in terms of accuracy between the priors and GBA, however results closer to GBA would be preferred. It should be noted that the largest error is in translation and that the rotation error is within similar bounds to the error we obtain from our inertial sensors.

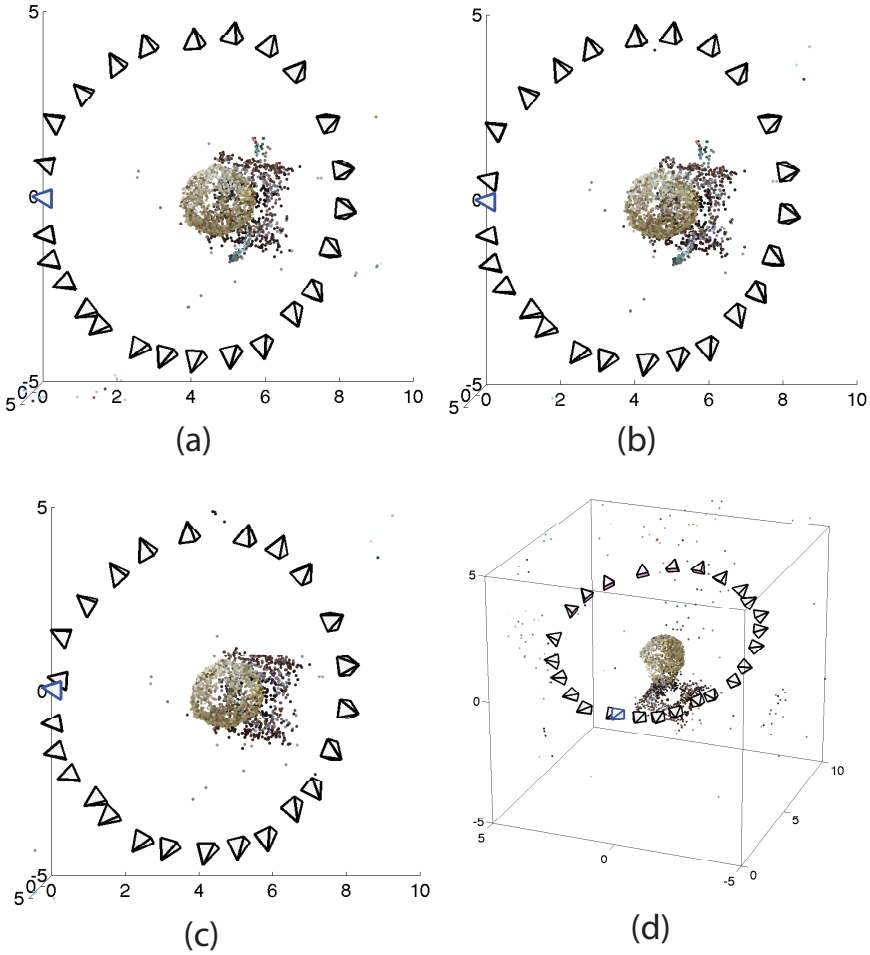


Figure 6.12: A visual comparison between the final camera poses obtained using our LBA algorithm versus the poses obtained using GBA. Figure (a) shows the initial camera positions prior to any optimisation, (b) is the result of running LBA and (c) is the result obtained from the GBA algorithm. Figure (d) shows the partial reconstruction of the bust from a different viewing angle in order to orientate the reader. The position of the first camera is shown in blue.

6.4.2 LBA Computational Efficiency

The main objective of the following test is to measure the running time of our algorithm with respect to that of GBA.

The LBA algorithm has a running time which is solely dependent on the number of features, as the number of frames is fixed to 3. On the other hand, the GBA algorithm grows with both the number of features and the number of frames. For this reason we evaluate the running time on a per iteration basis as this provides a truer reflection of the execution time.

The test was conducted by timing the optimisation over the bust dataset to ensure the number of features per frame was constant between both algorithms.

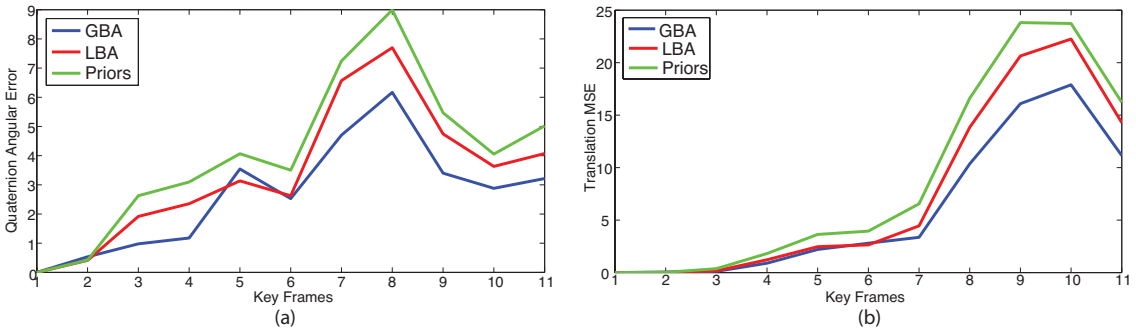


Figure 6.13: The quaternion angular difference (a) and translational MSE (b) comparing LBA and GBA to ground truth data obtained from the Vicon system.

The test was run ten times and the execution time was averaged over these runs. The results of this test are described in table 6.2.

Alg.	# Bundles	Time per bundle	# Iterations	Time per iteration
GBA	1	22.53s	27	0.834s
LBA	23	1.93s	14	0.138s

Table 6.2: Comparison of the computational efficiency of our LBA algorithm compared to that of GBA. The algorithm running times are worked out to an average per iteration time for fair comparison. All times are given in seconds.

It is clear from table 6.2 that our LBA algorithm is significantly faster than GBA when compared on a per iteration and per bundle level. However, it takes approximately double the time to complete the entire optimisation over all the frames. This behaviour is acceptable though, as the per bundle timing is more important in order to ensure that the optimisation completes before the next key frame is added.

6.4.3 Localisation Accuracy

As our localisation algorithm is used to determine when the system requires another key frame it is important that the algorithm is accurately able to determine the translation between frames. Furthermore, having an accurate localisation algorithm will reduce drift, jitter and excess computation in the pipeline. As the detection of a key frame relies on a significant change in pose occurring we use the output of the localisation algorithm to determine when a new key frame should be captured. Capturing too many key frames can lead to an increased computational load while capturing too few will cause the system to fail.

We perform the test by manually selecting two key frames from dataset 2, and using the 8-point algorithm and RANSAC to initialise 3D points and a set of tracking points. We then track these points from the second key frame for 60 frames, until the

point which we believe another key frame should be captured. We determine the pose of the device at each of these frames using the tracked points, their corresponding 3D points and inertial pose estimates obtained from our EKF.

As the pose from our localisation algorithm is used to determine when we require key frames it is important that the result is accurate enough to be able to determine when a large enough baseline has occurred. Furthermore, it is important for the localisation to be accurate in order to allow it to be used in augmented reality systems. For this reason we compare the pose of our localised cameras to the pose obtained by running GBA on the entire set of cameras and 3D points which we use for localisation. The results of these tests can be seen in figures 6.14.

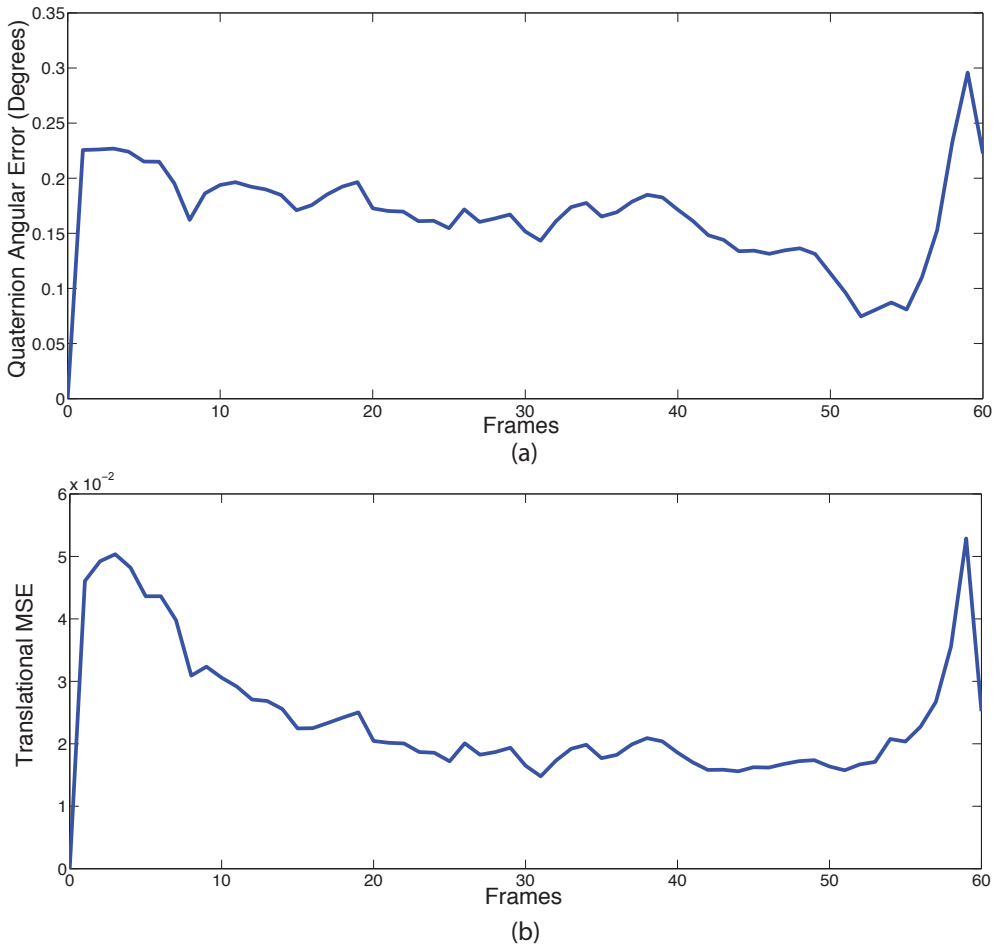


Figure 6.14: The quaternion angular difference (a) and translational MSE (b) when comparing the pose of our 60 camera poses to those obtained from GBA on the same dataset.

It can be seen in figure 6.14 that our localisation algorithm provides a very good approximation to the GBA poses.

6.4.4 Localisation Robustness

The accuracy of our localisation algorithm is very much dependent on the quality and accuracy of the feature correspondences and the accuracy of the 3D map points. The reason for this is that the localisation algorithm determines a frame's pose based upon 2D-3D correspondences between the tracked features and the 3D map points. As we do not check the accuracy of our feature tracker's output it is important that our localisation algorithm is robust against outliers.

The objective of the following test is to determine how robust our algorithm is to outliers in the feature correspondences. We perform this test using two implementations of our localisation algorithm. The first uses a standard L_2 error function and the second is the full implementation of our localisation algorithm, which makes use of a Cauchy M-estimator. Using the same approach and dataset as in our accuracy tests, we can test the effect of the M-estimator on the system's performance. We then run a series of tests where we add additional outliers to the dataset. As outliers are assumed to have no structure we add these outliers by creating correspondences between randomly selected image points. We summarise these results in figures 6.15 and 6.16.

Figure 6.15 demonstrates how sensitive the L_2 error function is to outliers, with a small number of outlying points leading to a large drift in our pose estimates. By comparing figures 6.16 and 6.15 we can see that the addition of an M-estimator into our localisation algorithm improves the overall robustness of our localisation. Furthermore, it is clear that even a large number of outliers has very little effect on the accuracy of the algorithm. Having a large number of outliers does however increase the computational expense of the optimisation.

6.4.5 Localisation Computational Expense

Our system attempts to localise the camera at frame rate, so it is important that our localisation algorithm is computationally efficient. The objective of the following test is to determine the computational efficiency of our algorithm.

This test was performed using the same dataset as the other localisation tests and the results were averaged. Our findings were that on average the localisation of a camera takes less than 5 iterations for the optimisation and has an average running time of 83ms. In the case where we do not initialise the optimisation with the inertial pose, we found that the average time to localise a frame increases to around 120ms and the average number of iterations increases to 11.

6.5 System Performance

Until now we have tested the performance of all the individual sub-systems, as well as smaller integrated sections such as the localisation which makes use of the EKF as well as our feature tracking algorithm. For this reason there is no need to

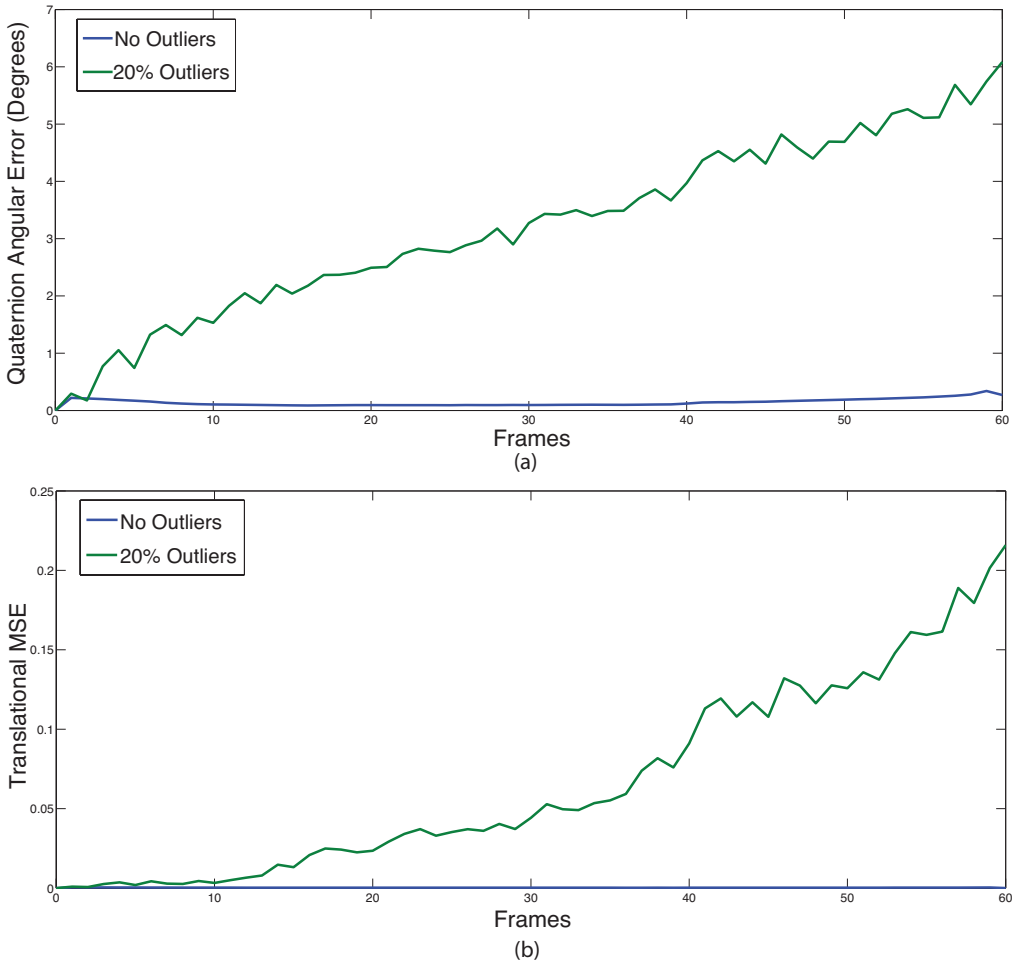


Figure 6.15: The quaternion angular difference (a) and translational MSE (b) when we expose our system to a number of incorrect feature correspondences when using the standard L_2 error function.

further quantitatively test the performance of the overall system, as we can draw conclusions based upon the performance of the individual sub-systems. Instead we rather present a qualitative overview of the system's performance.

The entire system was found to function with a reasonable accuracy on the condition that the motion remains slow. Under fast motion our system was found to fail, with the main point of failure being the feature tracker which cannot detect and track a sufficient number of features due to severe motion blur in the image frames. While our tracker can recover after bad frames have occurred, it can do so only if the rotational component of the motion is significantly larger than the translational component.

The computational efficiency of the system was found to be such that tracking and localisation can happen at approximately 3 frames per second in the presence of a moderate amount of features (60 features on average). The main bottlenecks

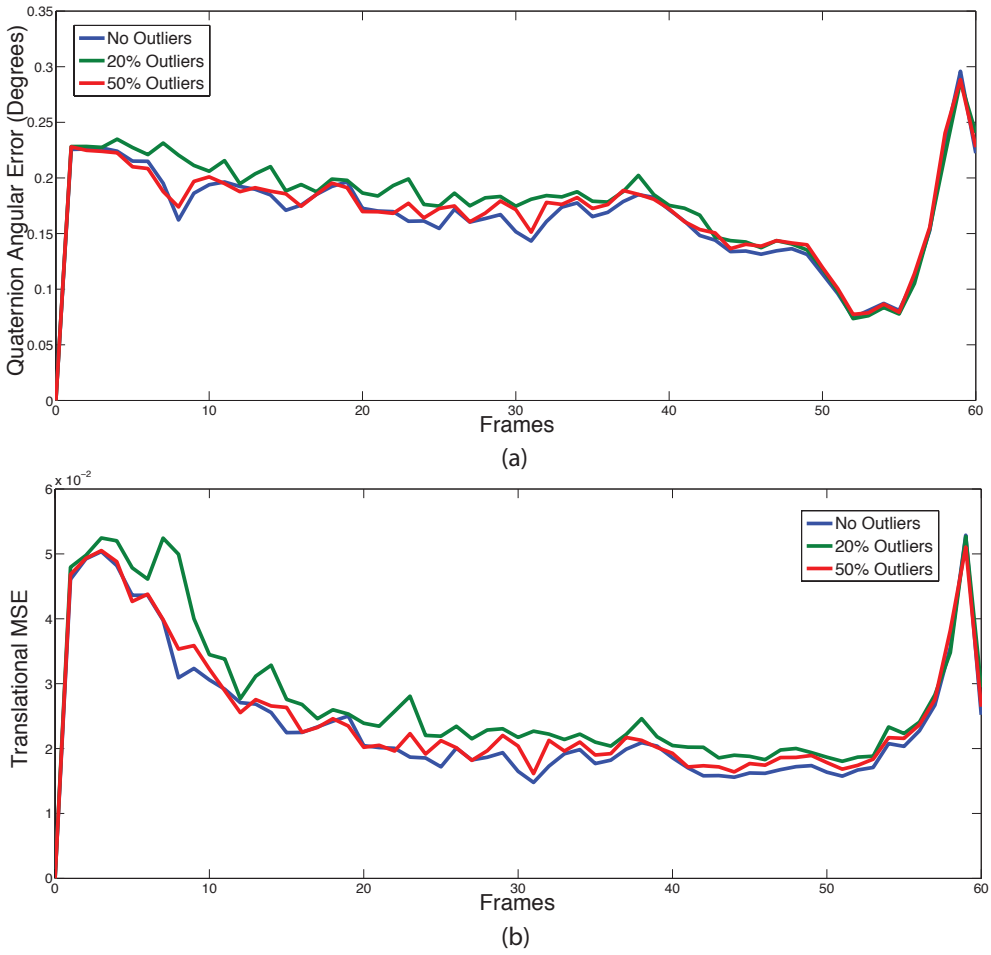


Figure 6.16: The quaternion angular difference (a) and translational MSE (b) when we expose our system to various levels of incorrect feature correspondences while using the Cauchy M-estimator.

were found to be the frame localisation and feature tracking stages of the pipeline.

Chapter 7

Conclusions and Future Work

The main objective of this thesis was to investigate how the additional sensors found on mobile devices can be used to increase the performance of a vision based pose estimation pipeline. For this investigation we made use of the low cost inertial sensors found on most modern mobile devices and sensor fusion techniques to estimate the device's attitude. This estimate was then used to assist visual feature tracking and visual pose estimation.

We focused on how the inclusion of inertial information can improve the overall system in terms of accuracy, robustness and computational efficiency. This was tested under the assumption of bounded motion dynamics and a rigid scene. We proposed four main sub-systems which we argue improve one or more of the three aspects above when compared to a purely visual pose estimation pipeline.

The first of these sub-systems is a quaternion based EKF to estimate the attitude of the device at 60Hz, from inertial sensor information only. We then proposed a hybrid feature tracking algorithm which uses inertial information and feature detection, as opposed to using feature detection and matching or optimisation based trackers. After that we proposed a reduced formulation of bundle adjustment on key frames, as an alternative to full scale global bundle adjustment which optimises over all points and cameras. Finally we proposed the use of optimisation and inertial pose estimation to localise the camera at frame rate rather than using epipolar geometry in a RANSAC framework.

In this chapter we make some concluding remarks about the various aspects of our system as well as present ideas for future work.

7.1 Conclusions

In chapter 1 we introduced the problem of monocular vision and structure from motion (SfM), and described the evolution of approaches taken towards solving this problem. This included a review of existing implementations of monocular SfM on computers as well as on mobile devices (section 1.3). We provided an overview of essential concepts and definitions to orient the reader in our problem, as well

as motivated the growing need for real-time, high performance pose estimation on mobile devices (section 1.2).

In chapter 2 we discussed and supported the use of quaternions as our choice for representing attitude. We further described the inertial sensors found on the most modern mobile devices and derived noise models for the sensors on the iPhone 5. During this investigation we found there to be a flaw in the magnetometer readings which we obtain from the iPhone 5. We investigated the flaw and determined that the magnetometer would report a discontinuity when it was influenced by an external magnetic disturbance. In section 2.2.4.2 we designed a filter, based on a Gaussian random walk, to correct for these discontinuities. This lead to increased accuracy of the magnetometer under static and small dynamic conditions, as seen in figure 6.3. Upon further investigation it was determined that dynamic motion and errors in the magnetometer's factory calibration lead to the same behaviour as magnetic disturbance, thus the filter could fail under these conditions as indicated in figure 6.4. Therefore we used magnetometer readings from a YEI-technologies sensor for the remainder of our tests. Near the end of the project it was brought to our attention that the erroneous magnetometer readings were caused by a faulty calibration at the factory. This calibration error was fixed in a later software update[1].

Next we derived a quaternion based extended Kalman filter (EKF) to perform sensor fusion and estimate the attitude of the device from the noisy inertial sensor readings, in section 2.3. Our EKF makes use of the TRIAD algorithm to synthesise an attitude measurement from the accelerometer and magnetometer readings. This synthesised measurement allowed us to use linear output equations and thus simplified the implementation and reduced the computational complexity of the filter. We found that our EKF implementation exhibits good performance under static and dynamic motion at an estimation rate of 60Hz. This was shown in figures 6.5 and 6.7 where we compared our pose estimates to the poses obtained from a very accurate Vicon motion capture system as well as a commercial IMU.

In chapter 3 we discussed two main approaches to determining feature correspondences between successive images, namely feature detection and matching, and feature tracking. We provided an overview of these approaches and discussed their suitability to implementation on a mobile platform. Following from this we introduced a hybrid feature tracking algorithm which makes use of feature detection and the continuous nature of video in order to track features between key frames. We made use of the Harris corner detector, as it can be implemented on the GPU and thus reduces the computational complexity of our feature tracker. We further integrated inertial information from the gyroscope into our feature tracking algorithm in order to compensate for large feature motion caused by inter-frame rotation. We found that the inclusion of inertial sensors into the feature tracking pipeline has a positive effect on the robustness of the feature tracker with little effect on the computational expense of the algorithm. This was found to be true for both fast and slow motions as shown in figures 6.8 and 6.9.

[1] `http://support.apple.com/kb/DL1691`

We found that while our feature tracking algorithm is significantly faster than the KLT tracker, it is still not efficient enough to enable true real-time tracking at 30Hz on a mobile phone. This can be seen in figure 6.11. Furthermore, we found that while our feature tracking algorithm appears to be less susceptible to drift than the KLT tracker, its accuracy is limited by the accuracy and number of features detected. Overall it was found to perform quite well under motions with constrained dynamics. As our tracking algorithm makes use of a tracking set which is determined by features in the previous key frame, we found that the number of 3D points is no longer a limiting factor. This was the case in the PTAM implementation [20] where all the 3D points were re-projected into every frame, and thus the 3D point count needed to be kept small.

Next we discussed how to use epipolar geometry to recover 3D information from point correspondences between pairs of images taken with a single camera, in section 4.1. This technique is used to initialise our system as described in section 5.6. In section 4.3 we discussed bundle adjustment and how it is used to improve camera pose estimates as well as the 3D points. We then presented our reduced formulation of global bundle adjustment (GBA) which optimises the cost function over three key frames instead of over the entire set of key frames. In order to maintain consistency and reduce drift only the pose of the most recent two key frames, and 3D points which were observed in more than one of the key frames, were allowed to be adjusted. Our local bundle adjustment (LBA) approach was shown to provide good accuracy (figure 6.13), while being significantly more computationally efficient than GBA, as seen in table 6.2. It should be noted that while the time to optimise the entire problem described in table 6.2 was almost double that of GBA, the iteration and bundle optimisation times were drastically reduced. This is preferred in key frame type systems where we need to ensure that the optimisation is complete by the time we add the next key frame, because each stage builds upon the accuracy of previous stages.

We also proposed an optimisation based algorithm for frame rate localisation. This algorithm makes use of tracked features and pre-existing 3D points in order to determine every pose. The cost function is the same as for GBA, however we only allow the camera's pose to be optimised. We further extended this approach to make use of our inertial attitude estimate to initialise the optimisation. It was found that this initialisation leads to an increased computational efficiency. We compared the accuracy of our camera localisation algorithm to the final GBA results over a set of 60 adjacent frames and presented the results in section 6.4.3. We found that our localisation algorithm performed well and would also enable accurate determination of key frames during tracking.

In general we found that while our localisation algorithm performed quite well it is still too computationally expensive for real-time implementation. We also determined that bad frames did not affect the localisation of future frames, as long as the tracker was able to recover features after a bad frame occurred. We further showed how the Cauchy M-estimator can be incorporated into the optimisation in

order to reduce the effects of outliers and bad tracking. These results were presented in figure 6.16.

Finally in chapter 5 we described how the four sub-systems can be combined into a full pose estimation pipeline which is coupled with the inertial sensors on a mobile device. We also discussed how to initialise the system and possible situations which will cause the system to fail.

Overall our implementations showed that the inclusion of inertial sensors into a vision based pose estimation pipeline can lead to significant improvements in terms of robustness and computational efficiency of the algorithms. However, it is difficult to draw conclusions about improvements in accuracy, although we can conclude that it is unlikely to negatively affect any aspect of the visual pipeline.

While our algorithms are still too computationally expensive to provide an enjoyable user experience on a mobile device they do provide an improvement on past attempts towards fast, accurate and robust monocular SfM on mobile devices. Furthermore, the computational power and quality of sensors on mobile devices are improving rapidly. The new range of mobile devices boast features such as general purpose programmable GPUs, better quality sensors and faster CPUs, all of which will greatly ease the implementation of real-time, high performance, pose estimation onto mobile platforms.

7.2 Future Work

The algorithms developed in this thesis serve as a proof of concept of how inertial sensors can be used to enhance vision based tracking, and the results were found to be promising. However, there are many areas which can still be improved and there is much research which can still be performed.

We found that the biggest limiting factor in our system is our feature tracking algorithm, as it relies entirely on the quality and accuracy of the feature detector and is too computationally expensive to run in real-time. This could be rectified by using a more robust and accurate feature detector such as SIFT and the mobile GPU to perform tracking. The tracking algorithm is highly parallelisable as each feature can be tracked independently of the others. We also found that our tracker exhibits problems in areas of small repeated texture, where multiple features would be tracked to the same point in the next image due to an overlap in their search areas and a small number of detected features. This problem might be rectified by adding a reverse tracking stage whereby features are tracked from the new frame back to the previous frame to ensure consistency.

If tracking is lost, or the quality of features degrades to such a point that the pose can no longer be accurately predicted, our system will fail and need to be re-initialised. For this reason a re-localisation algorithm needs to be implemented. This can be achieved through the use of downscaled versions of each key frame which are compared to the current frame. The pose of the downscaled key frame which best corresponds to a downscaled version of the current frame is used to re-initialise

the system. This approach was shown by Klein and Murray to work reliably with a very low overhead [19].

Furthermore, we do not make use of the uncertainty in attitude which is calculated as part of our EKF. This uncertainty could be used as part of the feature tracking algorithm in order to scale the size of the search area according to the uncertainty of the pose estimate. Furthermore, we could use this uncertainty in the localisation stage to determine if the attitude needs to be optimised. If we have a small uncertainty then we theorise that it is not necessary to optimise over the full pose, but rather only over the translation.

We may also consider adding the quaternion metric to our bundle adjustment and localisation cost function such as to bound the optimisation to an attitude which is near to our EKF estimate. This will prevent cases whereby bad frames or feature tracks cause the optimisation to converge to a pose far from the actual pose. It can even be coupled with a filter which flags incorrect pose estimates based on the requirement for smooth, continuous motion.

Future work may also include integrating our pose estimation pipeline into larger systems such as a through-the-window type augmented reality application, or a 3D reconstruction application. Doing so will add robustness to these systems and will provide an improvement to the overall performance. Furthermore, it is hypothesised that for an augmented reality (AR) system we can probably use the inertial pose directly, and only use video data to estimate translation. The reason for this is that a user is unlikely to notice a small error in rotation when the scene is sufficiently far away from the camera, as is the case in most AR applications.

Finally, we may consider incorporating scale prediction into our system by using the inertial sensors to predict scale. This will allow for true metric pose estimation, which is an enabling technology toward true, metric 3D reconstruction. Furthermore, if we could accurately estimate scale and translation we can re-project the 3D points onto the image plane and use these re-projected points to determine feature correspondences. We theorise that this approach will greatly improve the accuracy and computational efficiency of our feature tracking algorithm, even in the presence of motion blur.

As there are a lot of adaptions and possibilities on how to integrate inertial sensors into a visual pipeline, there are many more possibilities for future research. Due to the fast rate at which mobile device performance is advancing, as well as the ever improving quality of inertial and visual sensors on these devices, research into the problem of mobile pose estimation is likely to have a promising future.

List of References

[1] Y. Aksoy, "Efficient Inertially Aided Visual Odometry towards Mobile Augmented Reality," Ph.D. dissertation, Middle East Technical University, 2013.

[2] S. Baker and I. Matthews, "Lucas-Kanade 20 Years On: A Unifying Framework," *International Journal of Computer Vision*, vol. 56, no. 3, pp. 221–255, Feb. 2004.

[3] H. Bay, A. Ess, and L. Van Gool, "SURF: Speeded Up Robust Features," in *Proceedings of the 9th European Conference on Computer Vision (ECCV)*. Springer Berlin Heidelberg, 2006, pp. 404–417.

[4] J.-L. Blanco-Claraco, "nanoFLANN," accessed 27 March 2014. [Online]. Available: https://github.com/jlblancoc/nanoflann

[5] J.-y. Bouguet, "Camera Calibration Toolbox for Matlab," accessed 21 Feburary 2013. [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib_doc/

[6] G. Bradski, "OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000, accessed 17 Janurary 2014. [Online]. Available: http://opencv.org/

[7] F. Bruno, S. Bruno, G. De Sensi, M.-L. Luchi, S. Mancuso, and M. Muzzupappa, "From 3D reconstruction to virtual reality: A complete methodology for digital archaeological exhibition," *Journal of Cultural Heritage*, vol. 11, no. 1, pp. 42–49, Jan. 2010.

[8] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary Robust Independent Elementary Features," in *Proceedings of the 11th European Conference on Computer Vision: Part IV (ECCV)*. Springer-Verlag, 2010, pp. 778–792.

[9] A. Canclini and M. Cesana, "Evaluation of low-complexity visual feature detectors and descriptors," in *18th International Conference on Digital Signal Processing (DSP)*. IEEE, 2013, pp. 1–7.

[10] A. J. 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, no. 6, pp. 1052–67, Jun. 2007.

[11] J. Diebel, "Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors," vol. 58, 2006.

[12] C. Harris and M. Stephens, "A Combined Corner and Edge Detector," *Proceedings of the Alvey Vision Conference 1988*, pp. 231–236, 1988.

[13] R. Hartley, "An Investigation of the Essential Matrix," G.E. CRD, Schenectady, NY, Tech. Rep., 1995.

[14] R. Hartley and A. Zisserman, *Multiple View Geometry*, 3rd ed. Cambridge: Cambridge University Press, 2006.

[15] J. Hedborg, "Motion and Structure Estimation From Video," Ph.D. dissertation, Linkoping University, 2012.

[16] J. Heinly, E. Dunn, and J. Frahm, "Comparative Evaluation of Binary Features," in *Proceedings of the 12th European Conference on Computer Vision (ECCV) - Volume Part II*. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 759–773.

[17] A. Honda, "Methods for Enhancing Corner Detection," 2010. [Online]. Available: http://www.math.ucla.edu/~wittman/hyper/Report_Alex.pdf

[18] M. Hwangbo, J.-S. Kim, and T. Kanade, "Inertial-aided KLT feature tracking for a moving camera," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Oct. 2009, pp. 1909–1916.

[19] G. Klein and D. Murray, "Parallel Tracking and Mapping for Small AR Workspaces," *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 1–10, Nov. 2007.

[20] ——, "Parallel Tracking and Mapping on a Camera Phone," in *2009 8th IEEE International Symposium on Mixed and Augmented Reality*. Washington, DC, USA: IEEE, Oct. 2009, pp. 83–86.

[21] C. Konvalin, "Technical Document Compensating for Tilt, Hard Iron and Soft Iron Effects," MEMSense, Tech. Rep., 2008.

[22] D. Kurz and S. Ben Himane, "Inertial sensor-aligned visual feature descriptors," in *2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jun. 2011, pp. 161–166.

[23] B. Larson, "GPUImage," accessed 6 June 2013. [Online]. Available: https://github.com/BradLarson/GPUImage

[24] S. Leutenegger, M. Chli, and R. Y. Siegwart, "BRISK: Binary Robust invariant scalable keypoints," in *2011 International Conference on Computer Vision*. IEEE, Nov. 2011, pp. 2548–2555.

[25] M. Lhuillier and L. Quan, "A quasi-dense approach to surface reconstruction from uncalibrated images." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 3, pp. 418–33, Mar. 2005.

[26] M. Li, B. H. Kim, and A. Mourikis, "Real-time motion tracking on a cellphone using inertial sensing and a rolling-shutter camera," in *2013 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2013, pp. 4712–4719.

[27] D. Lowe, "Object recognition from local scale-invariant features," in *7th IEEE International Conference on Computer Vision*. IEEE, 1999, pp. 1150–1157 vol.2.

[28] C. Lu, G. Hager, and E. Mjolsness, "Fast and Globally Convergent Pose Estimation from Video Images," *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 22, no. 6, pp. 610–622, 2000.

[29] B. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," in *7th International Joint Conference on Artificial Intelligence (IJCAI) - Volume 2*, Vancouver, BC, Canada, 1981, pp. 674–679.

[30] J. Marins, E. Bachmann, R. McGhee, and M. Zyda, "An extended Kalman filter for quaternion-based orientation estimation using MARG sensors," in *2001 IEEE/RSJ International Conference on Intelligent Robots and Systems.*, vol. 4. IEEE, 2003, pp. 2003–2011.

[31] D. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the Society for Industrial & Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[32] D. Mount, N. Netanyahu, and J. Moigne, "Efficient algorithms for robust feature matching," *Journal of Pattern Recognition*, vol. 32, pp. 17–38, 1999.

[33] A. Mulloni, M. Ramachandran, G. Reitmayr, D. Wagner, R. Grasset, and S. Diaz, "User friendly SLAM initialization," in *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, Oct. 2013, pp. 153–162.

[34] R. A. Newcombe and A. J. Davison, "Live dense reconstruction with a single moving camera," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, Jun. 2010, pp. 1498–1505.

[35] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "DTAM: Dense tracking and mapping in real-time," in *2011 International Conference on Computer Vision*. IEEE, Nov. 2011, pp. 2320–2327.

[36] R. P. Pflugfelder, "A comparison of visual feature tracking methods for traffic monitoring," *ÖGAI Journal*, vol. 19, no. 4, pp. 15–22, 2000.

[37] M. Pollefeys and L. Van Gool, "Visual modeling with a hand-held camera," *International Journal of Computer Vision*, vol. 59, no. 3, pp. 207–232, 2004.

[38] M. Pollefeys, L. Van Gool, M. Vergauwen, K. Cornelis, F. Verbiest, and J. Tops, "Video-to-3D," vol. 34, pp. 1–12, 2002.

[39] L. Porzi, E. Ricci, T. A. Ciarfuglia, and M. Zanin, "Visual-inertial tracking on Android for Augmented Reality applications," *2012 IEEE Workshop on Environmental Energy and Structural Monitoring Systems (EESMS)*, pp. 35–41, 2012.

[40] S. J. D. Prince, *Computer Vision: Models, Learning, and Inference*, 1st ed. New York, NY, USA: Cambridge University Press, 2012.

[41] A. Ranganathan, "The Levenberg-Marquardt Algorithm," in *Tutoral on LM Algorithm*, June 2004, pp. 1–5.

[42] E. Rosten and T. Drummond, "Fusing Points and Lines for High Performance Tracking," in *10th IEEE International Conference on Computer Vision (ICCV'05) Volume 1*. IEEE, 2005, pp. 1508–1515 Vol. 2.

[43] ——, "Machine learning for high-speed corner detection," in *Proceedings of the European Conference on Computer Vision (ECCV)*. Department of Engineering: Cambridge University, 2006, pp. 1–14.

[44] J. Shi and C. Tomasi, "Good features to track," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 593–600, 1994.

[45] I. Skog and P. Händel, "Calibration of a MEMS Inertial Measurement Unit," in *XVII IMEKO World Congress*, Rio de Janeiro, 2006, pp. 17–22.

[46] H. Strasdat, J. M. M. Montiel, and A. J. Davison, "Real-time monocular SLAM: Why filter?" in *2010 IEEE International Conference on Robotics and Automation*. IEEE, May 2010, pp. 2657–2664.

[47] R. Szeliski, *Computer Vision: Algorithms and Applications*, 1st ed. Springer-Verlag New York, Inc., 2010.

[48] P. Tanskanen, K. Kolev, L. Meier, F. Camposeco, O. Saurer, and M. Pollefeys, "Live Metric 3D Reconstruction on Mobile Phones," in *2013 IEEE International Conference on Computer Vision*. IEEE, Dec. 2013, pp. 65–72.

[49] C. Tomasi and T. Kanade, "Detection and Tracking of Point Features," *International Journal of Computer Vision*, 1991.

[50] B. Triggs, P. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle Adjustment - A Modern Synthesis," in *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*. London, UK: Springer-Verlag, 2000, pp. 298–372.

[51] S. Umeyama, "Least-Squares Estimation of Transformation Parameters Between Tow Point Patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 4, pp. 376–380, 1991.

[52] A. van den Hengel, B. Dick, Anthony Thormählen, Thorsten Ward, and P. H. S. Torr, "VideoTrace: Rapid Interactive Scene Modelling from Video," *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3, 2007.

[53] L. Vicci, *Quaternions and Rotations in 3-Space: The Algebra and Its Geometric Interpretation*. Chapel Hill, NC, USA: University of North Carolina at Chapel Hill, 2001.

[54] T. von Marcard, "Design and implementation of an attitude estimation system to control orthopedic components," Ph.D. dissertation, Chalmers University of Technology, 2010.

[55] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," *In Practice*, vol. 7, no. 1, pp. 1–16, 2006.

[56] Z. Zhang, "Parameter estimation techniques: A tutorial with application to conic fitting," *Image and Vision Computing*, vol. 15, no. 1, pp. 59–76, 1997.

[57] ——, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 1998, no. 11, pp. 1330–1334, 2000.

[58] M. Zuliani, *RANSAC for Dummies*, draft ed. GNU Free Document License, 2008.