Applying Augmented Reality to Outdoors Industrial Use

Master's Thesis
University of Turku
Department of Information Technology
Software Engineering
2016
Mikko Forsman

Supervisors:
Jukka Arvo
Jouni Smed

Acknowledgements

This thesis was conducted as a part of the MARIN2 project (Mobile Mixed Reality Applications for Professional Use) funded by Tekes (The Finnish Funding Agency for Innovation) in Technology Research Center of University of Turku. The project was carried out in collaboration with partners; Defour, Destia, Granlund, Infrakit, Integration House, Lloyd's Register, Nextfour Group, Meyer Turku, BuildingSMART Finland, Machine Technology Center Turku and Turku Science Park.

UNIVERSITY OF TURKU
Department of Information Technology

MIKKO FORSMAN: Applying Augmented Reality to Outdoors Industrial Use

Master's Thesis, 72 p., 0 app. p.
Software Engineering
April 2016

Augmented Reality (AR) is currently gaining popularity in multiple different fields. However, the technology for AR still requires development in both hardware and software when considering industrial use. In order to create immersive AR applications, more accurate pose estimation techniques to define virtual camera location are required. The algorithms for pose estimation often require a lot of processing power, which makes robust pose estimation a difficult task when using mobile devices or designated AR tools. The difficulties are even larger in outdoor scenarios where the environment can vary a lot and is often unprepared for AR.

This thesis aims to research different possibilities for creating AR applications for outdoor environments. Both hardware and software solutions are considered, but the focus is more on software. The majority of the thesis focuses on different visual pose estimation and tracking techniques for natural features.

During the thesis, multiple different solutions were tested for outdoor AR. One commercial AR SDK was tested, and three different custom software solutions were developed for an Android tablet. The custom software solutions were an algorithm for combining data from magnetometer and a gyroscope, a natural feature tracker and a tracker based on panorama images. The tracker based on panorama images was implemented based on an existing scientific publication, and the presented tracker was further developed by integrating it to Unity 3D and adding a possibility for augmenting content.

This thesis concludes that AR is very close to becoming a usable tool for professional use. The commercial solutions currently available are not yet ready for creating tools for professional use, but especially for different visualization tasks some custom solutions are capable of achieving a required robustness. The panorama tracker implemented in this thesis seems like a promising tool for robust pose estimation in unprepared outdoor environments.

Keywords: augmented reality, tracking algorithms, pose estimation, natural features

TURUN YLIOPISTO
Informaatioteknologian laitos

MIKKO FORSMAN: Applying Augmented Reality to Outdoors Industrial Use

Diplomityö, 72 s., 0 liites.
Ohjelmistotekniikka
Huhtikuu 2016

Lisätyn todellisuuden suosio on tällä hetkellä kasvamassa usealla eri alalla. Saatavilla olevat ohjelmistot sekä laitteet eivät vielä riitä lisätyn todellisuuden soveltamiseen ammattimaisessa käytössä. Erityisesti posen estimointi vaatii tarkempia menetelmiä, jotta immersiivisten lisätyn todellisuuden sovellusten kehittäminen olisi mahdollista. Posen estimointiin (laitteen asennon- sekä paikan arviointiin) käytetyt algoritmit ovat usein monimutkaisia, joten ne vaativat merkittävästi laskentatehoa. Laskentatehon vaatimukset ovat usein haasteellisia varsinkin mobiililaitteita sekä lisätyn todellisuuden laitteita käytettäessä. Lisäongelmia tuottaa myös ulkotilat, jossa ympäristö voi muuttua usein ja ympäristöä ei ole valmisteltu lisätyn todellisuuden sovelluksille.

Diplomityön tarkoituksena on tutkia mahdollisuuksia lisätyn todellisuuden sovellusten kehittämiseen ulkotiloihin. Sekä laitteisto- että ohjelmistopohjaisia ratkaisuja käsitellään. Ohjelmistopohjaisia ratkaisuja käsitellään työssä laitteistopohjaisia ratkaisuja laajemmin. Suurin osa diplomityöstä keskittyy erilaisiin visuaalisiin posen estimointi tekniikoihin, jotka perustuvat kuvasta tunnistettujen luonnollisten piirteiden seurantaan.

Työn aikana testattiin useita ratkaisuja ulkotiloihin soveltuvaan lisättyyn todellisuuteen. Yhtä kaupallista työkalua testattiin, jonka lisäksi toteutettiin kolme omaa sovellusta Android tableteille. Työn aikana kehitetyt sovellukset olivat yksinkertainen algoritmi gyroskoopin ja magnetometrin datan yhdistämiseen, luonnollisen piirteiden seuranta-algoritmi sekä panoraamakuvaan perustuva seuranta-algoritmi. Panoraamakuvaan perustuva seuranta-algoritmi on toteuteutettu toisen tieteellisen julkaisun pohjalta, ja algoritmia jatkokehitettiin integroimalla se Unity 3D:hen. Unity 3D-integrointi mahdollisti myös sisällön esittämisen lisätyn todellisuuden avulla.

Työn lopputuloksena todetaan, että lisätyn todellisuuden teknologia on lähellä pistettä, jossa lisätyn todellisuuden työkaluja voitaisiin käyttää ammattimaisessa käytössä. Tällä hetkellä saatavilla olevat kaupalliset työkalut eivät vielä pääse ammattikäytön vaatimalle tasolle, mutta erityisesti visualisointitehtäviin soveltuvia ei-kaupallisia ratkaisuja on jo olemassa. Lisäksi työn aikana toteutetun panoraamakuviin perustuvan seuranta-algoritmin todetaan olevan lupaava työkalu posen estimointiin ulkotiloissa.

Asiasanat: lisätty todellisuus, seuranta-algoritmit, posen estimointi, luonnolliset piirteet

Contents

Chapter 1

Introduction

Augmented reality (AR) applications today are close to becoming a common tool for many tasks. Currently, most augmented reality applications cannot be considered as professional tools, as they are mostly used for entertainment, advertising and navigation.

Majority of the current problems have to do with accurate *pose estimation*. Pose estimation means the estimation of the position and orientation of the device to determine where the augmented content should be rendered. Pose estimation technologies still have major issues in reaching the required accuracy levels for professional use, especially in outdoors scenarios. Thanks to the rapid advances in hardware of mobile devices, AR applications are becoming more and more common. Better hardware is important since pose estimation technologies often demand a lot of processing power. Hardware improvements also include higher quality cameras as well as increased number of sensors. Since the prices of mobile devices are also dropping, it is easy to see that the potential for AR applications being used in everyday life is rising.

This thesis is structured as follows: This chapter explains the problems that the thesis tries to solve. Chapter 2 describes augmented reality as a term as well as its current and potential use cases. Chapter 3 contains related material to the systems implemented in the thesis. Chapter 4 discusses different pose estimation techniques. Chapter 5 describes different tools that can be used for the creation of AR applications. Chapter 6 presents the

software that was developed during the thesis. Chapter 7 presents the results achieved by the tools in the previous chapter. Chapter 8 contains the conclusions of the thesis as well as a discussion about the future of the projects in the thesis. The future of AR in general is also discussed.

1.1 The problem

Augmented reality has a large potential for multiple tasks in different industrial use cases. However, in order to be useful these applications must overcome the current issues that mostly have to do with estimating the pose of the user or device with sufficient accuracy, especially in outdoor scenarios when no prior (visual) data of the area is known. A large number of different solutions for visual pose estimation exist, but they are still often limited by high hardware requirements. Even if the hardware is the best possible available, visual pose estimation still cannot function in every situation. The algorithms required for accurate pose estimation can be very demanding, since a large number of camera frames needs to be processed every second. In order to make the augmentation look immersive things such as shadows and occlusion should also be taken into account. Occlusion happens, when the augmented object should appear to be behind a real object. Determining when the occlusion should happen is a very time consuming and difficult task. The shadows on the augmented object and the shadows cast by the augmented object are also very difficult to determine. Alongside with the rendering itself, solving these problems further increase the hardware requirements. This thesis aims at surveying different possibilities for creating augmented reality applications that can be used outdoors, as well as testing the usability of different methods in practice. The main focus of the thesis is on using visual data for pose estimation, but applications for using data from other sensors of the device are also briefly considered.

The majority of augmented reality applications today are based on monocular (single RGB camera) *visual tracking*. The term tracking means determining the movement and

position of some object or feature in the images from the camera. Rapidly changing environments and varying lightning conditions can make visual tracking a difficult task. Because of the nature of outdoor scenarios both of the aforementioned issues are present most of the time, which often makes visual tracking unreliable, if it is used as the only pose estimation technique. However, some visual tracking types that require no prior knowledge of the area can be useful in the outdoor scenarios, especially when combined with different sensors of mobile devices. Different types of visual tracking algorithms are discussed in more depth in Chapter 3.

Another common pose estimation technology is based on using *Inertial Measurement Units* (IMUs) of mobile devices. These include sensors such as magnetometers, gyroscopes and accelerometers. The issue with IMUs is the accumulation of *drift*. Drift happens due to small errors in determining the movement, which can cause the measured results deviate from the real results over time. However, combining these IMUs with some visual tracking is currently the most promising method for creating a robust AR experience. The pros and cons of different sensors are discussed in Section 5.1.2.

The proposed solution is to create a system which relies mainly on visual tracking, does not require any prior visual knowledge (i.e. markers), and uses the sensors of the mobile device to support the tracking. The proposed systems do not provide any solution for estimating the absolute orientation of the device except the sensor fusion algorithm which is presented in Section 6.2. To determine the position of the user in the world coordinates with sufficient accuracy more research is required to improve the accuracy of the *Global Positioning System* (GPS) and magnetometer or some similar positioning system. Improvements of positioning systems however falls out of the scope of this thesis.

There are some predefined requirements for tracking system: the accuracy should be sufficient enough so that there are no visually noticeable inaccuracies in the tracking quality (i.e. the augmented object should seem to stay still in the real world). The system should also be relatively simple to be runnable on modern mobile devices with at least 15

frames per second. In order to be easily integrated into larger systems, the tracker should also be usable in Unity 3D game engine [1].

The research questions of this thesis can be summarized as the following:

1. What are the existing technologies that are currently usable for creating mobile augmented reality solution for outdoor use?
2. Can these technologies be used to build a robust tracking system that can fill out the previously defined requirements during the thesis?

Chapter 2

Augmented reality

Augmented reality means the augmentation of some virtual content, such as 3D models, videos or images on top of an image of the real world. An example of a simple AR application, where a 3D model is augmented on top of an image marker, is shown in Figure 7.2.

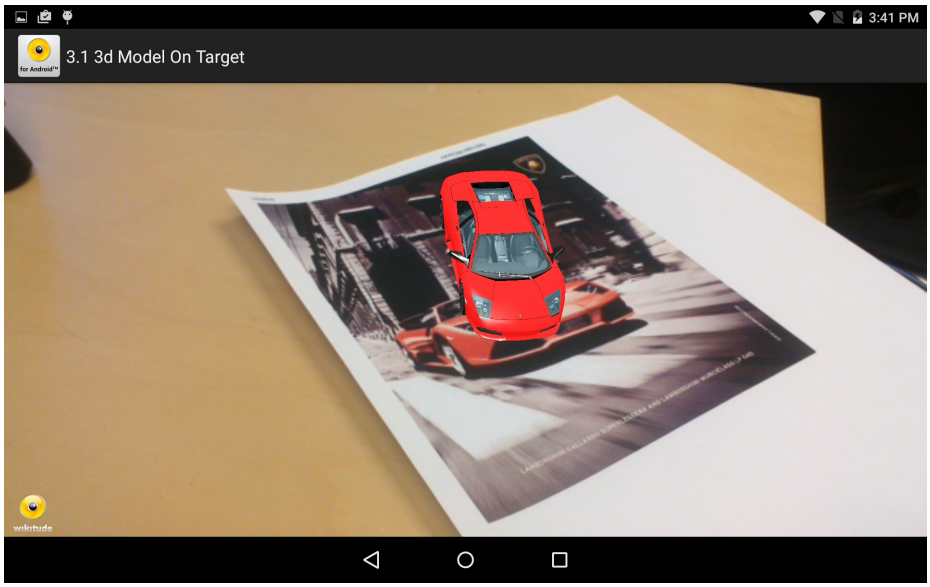

Figure 2.1: Example of Augmented Reality from Wikitude demo application

Most of the time this means using the screen of some mobile device (tablets or smartphones) for the augmentation. It is also possible to use some designated AR devices such

as Microsofts Hololens [2]. However, AR can also be applied through, for example, auditory content [3] or even smells [4]. In order to augment some visual content to the user, the pose of the user (both rotation and translation in the world coordinates) must be known in order to augment the content to its correct position. This estimated pose can be computed with a wide variety of techniques, including tracking of markers and estimating the pose using inertial measurement units of a mobile device. AR is often defined using the Milgram's Reality-Virtuality continuum [5] (see Figure 2.2), which presents the range from the real world to completely virtual world. It places AR in the area of Mixed Reality (MR), between reality and augmented virtuality. *Diminished reality* [6], in which some content is removed from the image of the real world (for example removing of a chair from a scene etc), is sometimes also considered as a part of the continuum.

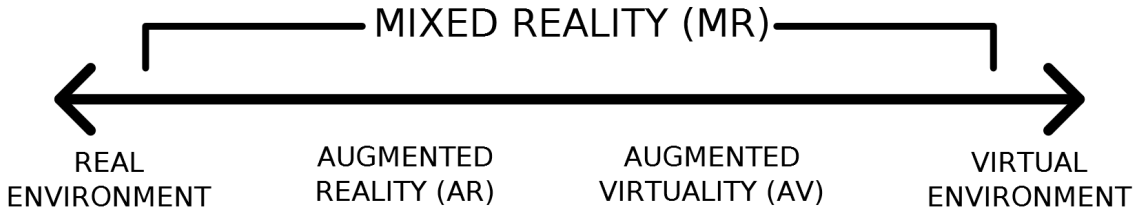

Figure 2.2: Milgram's Reality-Virtuality Continuum

2.1 Current state of AR

The current state of AR technology is still not ready to be used as professional tools. AR is still lacking some technology, such as good enough virtual glasses or helmets. Also, the accuracy of different pose estimation methods is not always sufficient, especially when visual tracking is used and prior visual knowledge of the area is not available. Some simple marker-based pose estimation technologies can already reach a very robust accuracy, but, unfortunately, marker-based tracking is not usable in larger scale scenarios. Marker-based tracking is problematic in large scale scenarios, since markers would have to be large and they would need to be placed in a large enough number of places accurately.

When using a more complex pose estimation system instead of tracking markers, the processing power requirements also grow, which is a challenge when applications need to run on mobile devices.

2.1.1 Current AR use cases

Currently, AR is mainly used on rough demonstration, gaming and marketing applications. For these types of applications, the requirements for the accuracy of tracking are not so high when compared to professional use cases, and these simpler applications can use simpler tracking techniques such as marker-based tracking.

For marketing purposes, AR can be a powerful tool due to often simple applications. For example, the user can scan the side of the box to view its contents in 3D, or possibly see some instructions on how to use the product inside. Since such applications do not require the exact position of the user in the world and the accuracy of the tracking is not that important, these applications can be simple to build. Multiple companies provide services for building marketing AR applications, and they can often be built with a simple toolset. For example, Wikitude [7] contains a drag-and-drop type tool for creating AR apps in addition to AR browser which can be used to view user created AR content. A good example of an AR marketing tool is TryLive's facial recognition tool [8], which allows users to try different types of eyeglasses using AR. For these kinds of applications, AR can already be used efficiently.

Another large field for AR is education. Augmenting content on top of a book or any other visual target is quite a simple task. Students can, for example, view videos regarding whatever they are currently studying, or see and interact with 3D models instead of plain images. Luckin and Fraser [9] tested the usefulness of AR in education and evaluated over 300 participants, and came to the conclusion that AR has large potential in educational tasks. For education, the current state of AR is already enough in order to create applications of sufficient quality. However, it can get expensive when every student

requires also a device to view the AR content. It is often difficult to find early adopters for a new technology, since the prices of required devices can be high. Especially in education, if a device is required for each student, the cost might be too expensive.

AR has applications in navigation as well. For example, Wikitude provides an AR navigation system for smartphones [10] that augments your route on top of the video feed from the road you are driving on to make it more obvious where you should turn (see Figure 2.3). The same concepts that can be used in outdoor navigation can also be applied for indoor navigation to point out different positions on the screen with higher accuracy than, for example, plain GPS or bluetooth navigation.

Figure 2.3: Wikitude AR navigation tool. Image taken from Wikitude AR navigation tool demo video [11]

A large potential for AR is also present in entertainment industry, mainly in gaming. Examples of AR games have been made using Microsoft's Hololens [2] [12] or Magic Leap [13]. However, the largest issue with AR gaming are the same as in industrial use: the quality of the augmentation (shadows, lightning and occlusion should match the real world) and tracking has to be extremely good for it to be visually appealing. Further-

more, current devices do not have enough computing power when the games become too complicated and have a large amount of things to track and render.

2.2 AR in industry

The main difference with marketing, navigation and simple visualization use cases and industrial use cases is that industrial use requires much higher accuracy in pose estimation and overall higher quality of tools. For tasks such as visualizing some part of a building or viewing maintenance instructions, the quality has to be higher since errors in augmentation can cause some major problems. For example, a problem in the building planning might go unnoticed or the maintenance of the tool could be done wrong, which can both be costly mistakes. For industrial use cases, the development of the pose estimation system is also often difficult: a large number of scenarios take place outdoors, where the environment is unprepared and can vary a lot due to weather conditions and moving objects. AR has currently a limited number of real use cases in industry, because AR is still quite immature and requires improvements in both hardware and software. Since in industrial use the accuracy of the tracking is much more important than in, for example, navigation or advertisement, it is much more difficult to create AR applications for these use cases. Moreover, industrial use often requires more complicated tracking systems, in which tracking simple markers is not good enough. The pros and cons of different tracking systems in outdoor use is discussed more in Chapter 4.

2.2.1 Current and potential use cases in industry

AR could potentially be used in a large number of different industrial cases, most intuitive being visualization tasks. These visualization applications could drastically reduce the number of tools that are currently required to keep track of the projects. The users could, for example, view how a building will look like when it is completed, and they

could ”see through” already built walls to locate different pipes and wirings. These different visualization tools could be useful in noticing problems with the plans earlier, for example, if different parts of the models are overlapped or do not fit together well. Visualization tools would also be helpful on the planning phase of different industrial tasks, since the plans could be viewed in the real world with ease. These applications would also work very well for demonstrating planned 3D models in real world before actually building the models. This thesis especially aims at creating a system which could be used for visualization of large objects such as buildings in their planned locations.

Another potential use case for AR in industry is different maintenance tasks. Workers who are unfamiliar with devices or vehicles often need someone more experienced to help them in maintenance tasks. However, by utilizing AR all the required instructions could be easily shown to the user on top of the actual device. These kinds of applications already exist: for example, AR-media has built I-Mechanic application [14], which allows users to do maintenance on their car without requiring any prior knowledge. Similarly to maintenance, AR could also be used as a tool for teaching how to use complex machinery that otherwise takes a long time to learn. In smaller scale scenarios, where the pose of the device can be estimated by tracking of markers, AR is already in a usable state. The problem with maintenance applications is often the requirement of hands-free tools. In order to view the instructions and simultaneously do the maintenance, AR-glasses or some similar tool is required.

Navigation can also be considered as an industrial application for AR. Besides more traditional navigation systems in vehicles, AR tools could be used to help users navigate in large industrial halls to find some certain piece of equipment. Especially indoor navigation could use AR. For example, the users could be shown the accurate location of a certain item in a warehouse.

Chapter 3

Related work

One of the first steps during this thesis was to study existing methods for outdoor AR applications. The focus was in finding algorithms that could answer the research questions of this thesis. This chapter presents work that was considered to be closely related to the introduced implementations in Chapter 6.

One of the most interesting articles about outdoor tracking is the article by Azuma et al. in 1998 [15], which claims that it is impossible to create an outdoors tracking solution without using multiple different tracking types, i.e. sensors and visual data. Even though the article is relatively old, it is still valid to some degree. Due to the uncertainties in an outdoors scenario, it is understandable that using plain visual tracking is not always possible. Visual tracking also suffers from motion blur caused by fast movements and changing lightning conditions, which can be compensated for using inertial measurement units of different mobile devices.

One system built for outdoor tracking was proposed by Menozzi et al. [16]. The paper proposes a tracker which uses inertial sensors whose data is combined with a visual tracker, and it is implemented as a standalone device. The visual tracker is based on searching for features in the geometry of the horizon line, and it requires a calibration step for finding the initial orientation of the device. The system proposed in the article is able to estimate both the position and the orientation of the device. The error in the

orientation was reported to be below 10 mrad from the real orientation when combining all the different tracking types presented in the publication.

The paper presented by Wagner et al. [17] (which is used as a reference for the panoramic tracking system implemented in Section 6.4) explains the use of panoramic images in tracking in both indoor and outdoor scenarios. The panorama maps generated by the system are high quality panoramic images. The idea of the system is to simultaneously create the panoramic image according to the estimated orientation of the device. This technique is very powerful since it does not accumulate any drift, other than the drift caused by errors in the map creation process. The system ran with a very good performance on mobile devices; the results presented on the article claim that the tracking of single frame completed in average of 15.2 milliseconds for mobile devices and 2.2 milliseconds for PC. The paper also reported that the amount of errors accumulated in the map was very small, and the horizontal errors in the panoramas were in the range of a single degree. This level of accuracy would be sufficient for the requirements of the system. The major downside of the system is that it is only capable of tracking the rotation of the device, but not the translation.

The panoramic tracking presented by Wagner et al. was also incorporated in to a tracking system by Schall et al. [18]. The system uses a Differential GPS (DGPS), magnetometer and and a barometer to determine the absolute position of the device. The orientation is calculated using an inertial measurement unit that composes of gyroscopes, magnetometers and accelerometers. A visual tracker (the previously mentioned panorama tracker) is also used in order to reset the drift from the inertial measurement unit. The biggest difference of the system in the paper and the solution proposed in the thesis is that the authors of the papers built a hand-held system from a scratch, where as the proposed solution of the thesis aims to be runnable on basic mobile devices. The conclusions of the paper were promising, as the Kalman filtered compass error stayed below a single degree combined with the visual tracker that could also reach a similar accuracy. The accuracy

of the DGPS system was also in a range of one meter, which is notably better than the raw GPS accuracy that is usually in a range of ten meters.

A recent research by Jing Li and Xiangtao Fan [19] proposes another method for outdoor tracking. The method uses 3D models of the buildings in a city for estimating the user's orientation. This type of tracking is possible since an increasing amount of different buildings have their 3D models available for anyone, and the increased processing power of mobile devices is becoming capable of tracking complex objects. The results show a tracking system that runs around 6 to 7 frames per second, which is not quite sufficient for a smooth user experience. The paper was published in 2014, so even with current hardware the fps rate would probably be a lot better. With some optimizations different types of 3D model trackings can be very helpful in outdoor scenarios.

An evaluation regarding wearable AR devices for outdoors uses was conducted by J Kerr et al. [20]. The paper presents that wearable AR devices still need development in different areas in order for them to be usable. The largest amount of issues are presented by the screens in the devices. The paper reported that the users had issues in seeing the screen in outdoor lightning conditions. Interaction with the device was also difficult, since gesture-based interaction caused the inputs to be interpreted wrongly or accidentally. The users also reported that the wearable should be less 'obvious', since now it was relatively intrusive. Some participants also reported that wearing the device for more than 10 minutes at a time felt uncomfortable. The findings presented in this paper are also a reason for why the proposed system uses a hand-held mobile device instead of some wearable AR device. The issues of wearable AR devices are discussed more in Section 5.1.3.

After studying scientific material related to the subject of the thesis, it became apparent that in order to create a robust system which does not require any prior knowledge, combination of different pose estimation systems is required. A large number of papers also propose building a system from scratch instead of using for example tablets or mobile phones. This is due to the fact that even five years back the hardware of mobile devices

was lot worse, and often did not include high quality IMUs. The quality of the cameras has also improved a lot. This lead to the conclusion that building a custom hardware solution is not required. During the thesis it also became apparent that any custom hardware was not needed.

Chapter 4

Visual pose estimation

The term *pose* is used to describe both the rotation and the translation of the used device in some coordinate system. This computation is known as a *6 degrees of freedom* (6DoF) pose estimation, when rotations around x, y and z axis as well as translation in all three axis are estimated. A simpler version of the pose estimation, known as *3 degrees of freedom* (3DoF) pose estimation, is also used in some cases, where we have either the knowledge of rotation or the knowledge of the translation. By combining different 3DoF systems, it is possible to achieve a 6DoF system.

Pose estimation is the most important part in creating visually appealing AR applications. If tracking is not robust or accurate, the immersion of the content being in the real world is easily broken, since the augmented content will be viewed in a wrong position or it might jump around. For estimating the position of the user, different sensory data (visual data and inertial measurement units) is used. In order to be a complete tracking solution, the pose must be estimated in 6DoF. Single tracking systems are sometimes only capable of estimating 3DoF (pure rotational or translational) pose, but these systems can always use some other method for estimating the remaining 3DoF.

Tracking the pose in AR applications can be split in to two categories: visual and non-visual tracking. Visual tracking is the most common tracking type for most of the applications, since it is often not important to augment the content to an absolute position

in the world, and there is not always a guarantee of all the required sensors existing. In these cases, the augmented content can be viewed on top of some pre-defined markers or previously scanned 3D objects. There are also tracking systems that use visual tracking without any prior knowledge such as *Simultaneous Localization And Mapping* (SLAM) [21] and the tracking systems explained in Section 6.3 and Section 6.4.

Tracking the environment using markers or other predefined visual data is a difficult task to accomplish outdoors. When considering large scale visualization scenarios, placing a large number of markers accurately is often too time consuming. Such amount of work would also mean that the usability of the system becomes poor. Visual tracking methods using pre-defined templates or markers are typically useless in an environment which varies a lot, such as work sites that have a large number of moving elements and varying lightning conditions. However, using markers or scans of some objects could be used in some smaller scale use cases such as viewing maintenance instructions for some tools.

Visual tracking methods other than marker-based tracking can be viable in outdoor industrial scenarios. These methods can create a map of the environment during the runtime of the application. One example of such systems is the aforementioned SLAM, which tracks the movements of the device in 6DoF by consecutively estimating the pose, and adding new features to the map when the device is moved to a position where it has not been before. There are also simpler approaches to create a trackable map of the area such as panorama tracking proposed by Wagner et al. [17]. Panorama tracking is used as a reference method for the solution proposed in Section 6.4. However, these types of tracking solutions have another issue, since they are sourceless (i.e their initial pose in the world is unknown). In order to be useful, knowledge of the initial starting position and orientation of the device is required. A lacking initial position can be computed, for example, by using magnetometer and GPS or some previously defined visual target.

4.1 Marker based tracking

Probably the simplest method for visual pose estimation is to track different types of markers. The simplest markers are black-and-white images consisting of a grid of squares that are either black or white (see Figure 4.1). However, markers do not have to be that simple, since any kind of images can be used. Most of the time black and white markers are practical to detect due to high contrast differences.

There are two major downsides in using markers in outdoor environments. Firstly, someone has to place a large number of markers around the scene to predefined positions in order for the system to work. Secondly, marker-based systems suffer a lot from varying lightning conditions (i.e. shadows), and, therefore, some camera setting calibration is required whenever the lightning is different. Markers also have to be pre-defined, whereas an optimal system should work without any prior knowledge of the area. For large scale visualization tasks, the required size of the markers becomes large. If the marker is very close to the camera and the augmented content is far behind, even small errors in the tracking can cause the augmented content to move around a large amount. Otherwise, the marker needs to be extremely large in order to be trackable from afar. It is also possible to improve the quality of marker based tracking by using multiple markers.

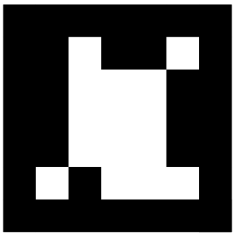

Figure 4.1: Example of a simple black and white marker

4.2 Natural feature tracking

The tracking methods used in the thesis are based on sparse optical flow and template matching, which both count on tracking natural features. *Natural feature tracking* detects and tracks features from the image that are easily distinguishable. The largest benefit of natural feature tracking is that it is not always necessary to know anything about the scene beforehand, which is a huge advantage in scenarios that change often. A large number of algorithms exists for both finding and tracking natural features. Especially in outdoors tracking using natural features, which are not defined beforehand, offers a large potential due to the problems of pre-defined markers presented in the previous section.

4.2.1 Finding trackable features

One of the most important things for natural feature tracking is locating the features that can be distinguished easily, since not every pixel can be tracked. For example, if a point in the image is selected that is located on a vertical edge, it can only be used to track the horizontal motion accurately. Figure 4.2 shows an example of a good and a bad feature: the red feature on the edge is difficult to track in horizontal direction, since a similar-looking position can be found anywhere in the edge. The feature on the corner, however, can be reliably tracked in both directions. The trackable features must be easy to detect, but almost as important is the performance of the feature finder. Some trackers rely on searching a large set of points at every frame, which can cause performance problems if the selected feature finder is not simple enough. A large number of different algorithms have been suggested for finding these features. These feature finders are often also called *corner detectors*, since the best trackable features in images are usually corner positions, since those points can be reliably tracked in both horizontal- and vertical directions.

One commonly used corner detector is Harris detector [22], which works by first applying a Harris edge detector on the image, and then processing each pixel to calculate

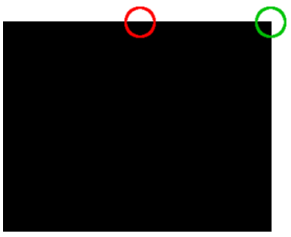

Figure 4.2: Example of good feature (green circle in the corner) and bad feature(red circle in the edge)

whether or not the edge pixel is a feature of sufficient quality. Harris detector computes a score for each pixel by calculating the difference of the pixel intensities in all directions. As a result, each pixel in the result image has score between 0 and 255. The higher the score, the better feature that pixel is.

The feature finder used in Section 6.3 uses a method proposed by Shi and Tomasi [23] based on the Harris detector. The difference to the original method is that the score of each pixel is calculated in a differently. This process is implemented in an OpenCV [24] library function goodFeaturesToTrack. The drawback of this method is that the speed is often not quite enough for realtime applications, especially if the features need to be searched for often and in large areas.

For the matching that is used in Section 6.4, the features are searched with FAST (Features from Accelerated Segment Test) algorithm by Rosten and Drummond [25] [26]. This algorithm is very useful in scenarios where a large number of features needs to be detected, and the method consumes very little processing power. FAST examines a circle around each pixel in order to figure out whether the pixel can be used for tracking according to a set threshold. The radius of the circle is 16 pixels. The pixel is considered a good enough corner if at least 12 neighboring pixels in the circle differ enough from the intensity of the center pixel. Figure 4.3 shows a sample image and a corner detected with the FAST algorithm.

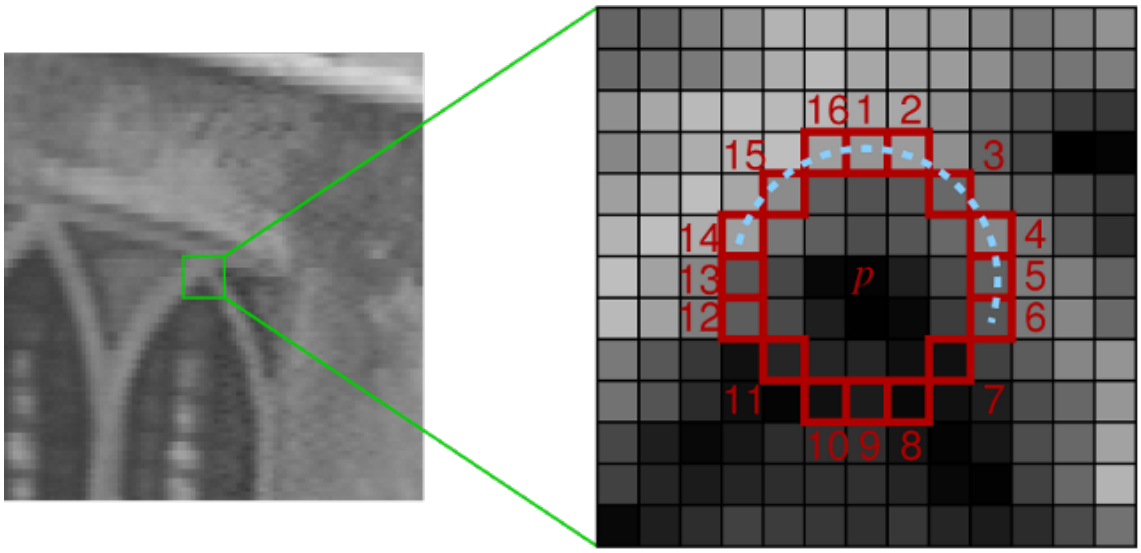

Figure 4.3: How FAST corners are calculated for each pixel. Sample image from Edward Rosten's web page [27]

Another common feature finder is the ORB (Oriented FAST and rotated BRIEF) [28] algorithm, which was also tested in the implemented systems. It is a feature finder based on FAST, which can also be used for extracting knowledge of the rotations of the features.

In this thesis, a test application was developed in order to compare different corner detectors to determine their performance as well as the quality of the features located by the detector. The result images are presented in figures 4.4 and 4.5. The performance for each detector was also measured in the first sample image on PC (Lenovo T440p laptop with Intel Core i7-4710Q CPU). The runtimes of each detector are present in Table 4.1. It is worth noting that the detector does not always need to be run on whole images. The source image resolution was 640x480 pixels, and the tests were based on the implementations available in OpenCV library.

4.2.2 Matching features

After a set of features have been selected for tracking, they have to be matched to estimate the movement between two frames. In order to do this, descriptors have to be extracted for

Table 4.1: Runtimes of corner detectors

Detector	Algorithm runtime
ORB	179ms
Shi and Tomasi	41ms
Harris	28ms
FAST	1ms

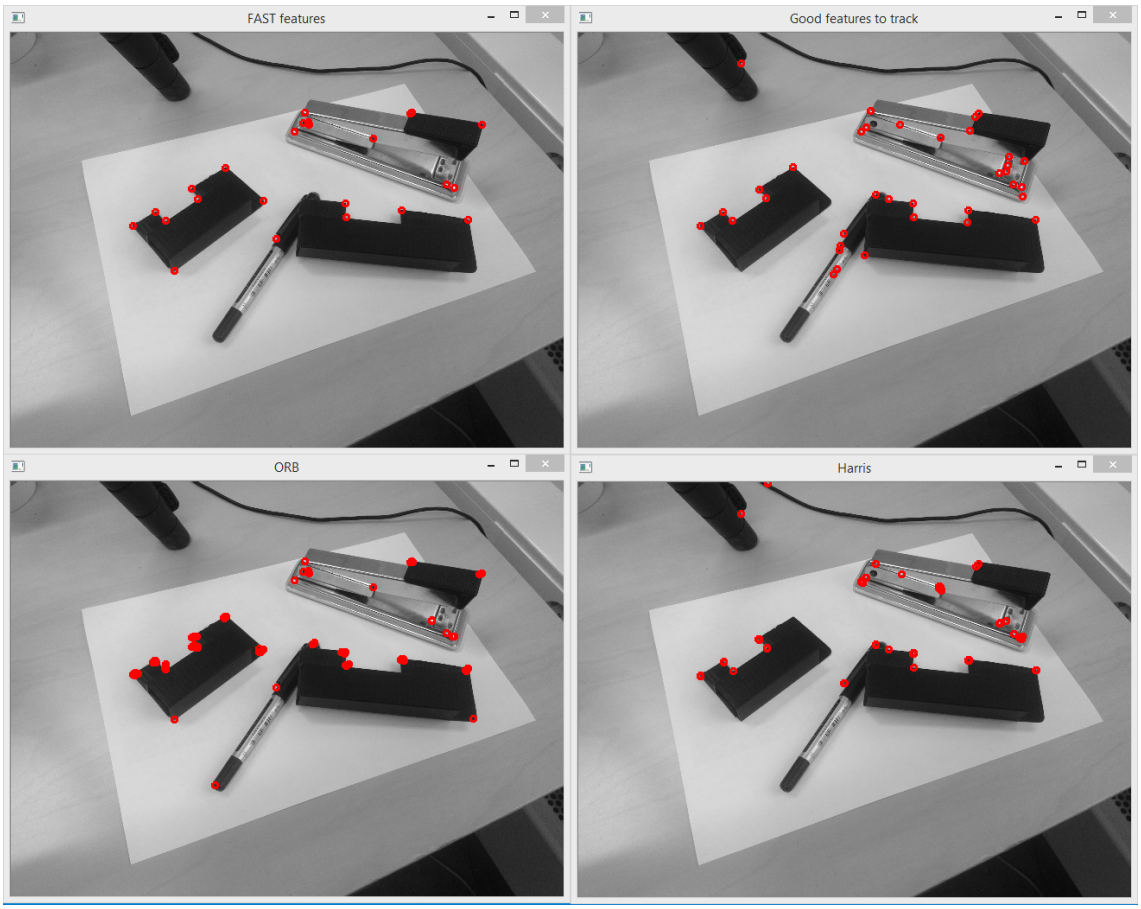

Figure 4.4: Test results for different corner detectors. Red circles are the detected features. Top left corner FAST detector, top right corner is Shi and Tomasi method, bottom left ORB and bottom right Harris detector.

each feature which will then be used for the matching. Descriptors are used to describe the area around the feature. For each frame, a set of features and their corresponding

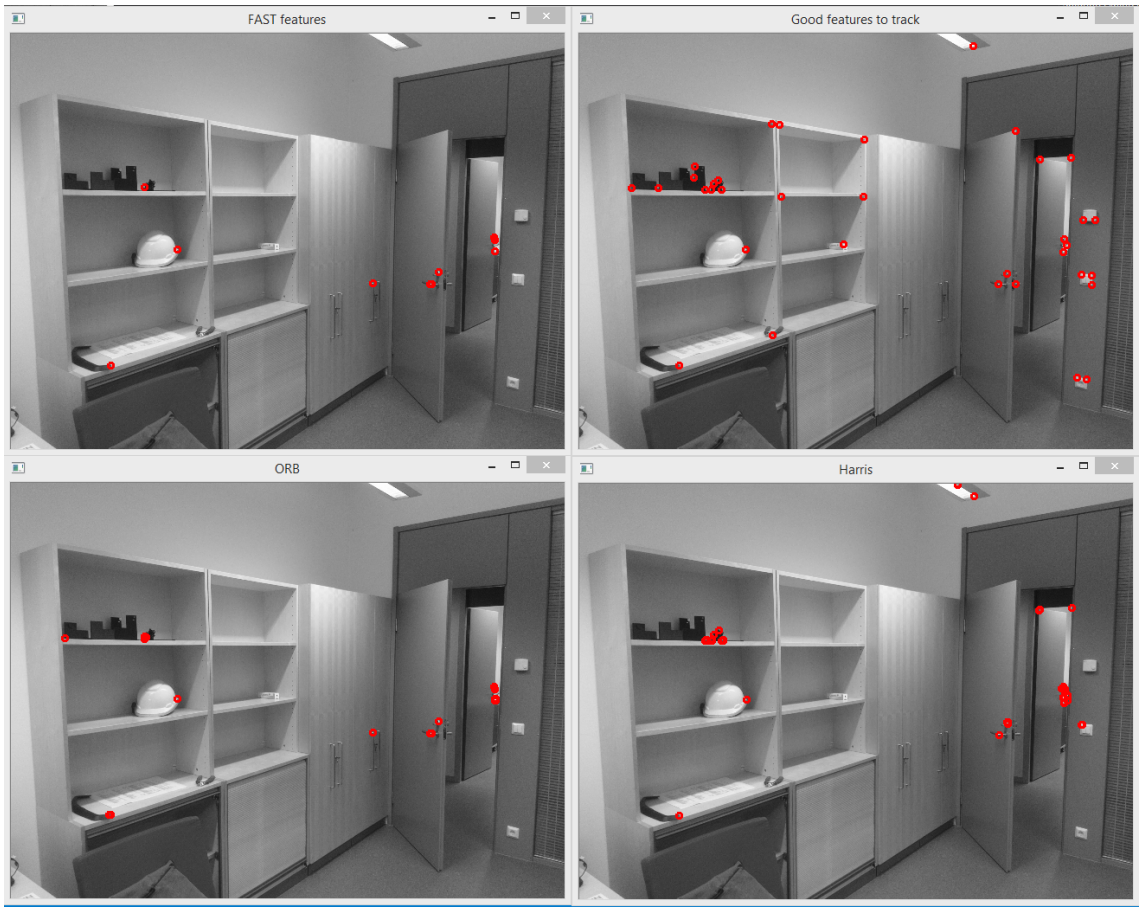

Figure 4.5: Test results for different corner detectors. Red circles are the detected features. Top left corner FAST detector, top right corner is Shi and Tomasi method, bottom left ORB and bottom right Harris detector.

descriptors are extracted from the image, and they are then matched against a pre-defined set of features. The pre-defined set of features could be a set from the previous frame or something else such as a set extracted from an object that should be tracked.

Template matching

In Section 6.4, a template matching is used to track the motion between successive frames. Template matching is one of the simplest methods for finding a certain area of an image from another image, so the descriptor can simply be considered as an image of the area around the feature. The matching happens simply by ”sliding” the template area across

the searchable image, and calculating a similarity threshold in each position using convolution. Due to the simple nature of this tracking type, the template matching algorithm also has its own problems. The method takes a lot of processing power to calculate the matching for large templates, and the method is not *scale or rotation invariant* (template matching cannot match the template, if the template is rotated or scaled differently than the original image). Therefore, the template matching requires additional methods for calculating the rotation and scaling. If the tracking system can somehow overcome these issues, template matching can be a useful tool for tracking. Figure 4.6 shows an example of template matching.

In some cases, template matching can be used in tracking applications. These scenarios usually require that the matched templates are small (preferably less than 10x10 pixels), and the areas they are matched against should also be small. Using template matching for pose estimation is possible, if the pose can be estimated with good accuracy prior to the matching or the movement between successive frames is small. Scale and rotation invariance issues are more difficult to overcome. If the application uses template matching for rotation and scale, each of the templates must be scaled and rotated to different positions and matched with the target image. However, rotating, scaling and matching each template can be very computationally intensive, since a large number of matches must be calculated for each template. Template matching can be used in tracking by, for example, extracting areas around detected features, and using those areas as templates.

SIFT

SIFT (Scale Invariant Feature Transform) [29] is a method used for both finding and matching keypoints. What makes SIFT special is its invariance to translation, scaling, rotation and even slight illumination changes. The basic idea of SIFT is to describe the found features as vectors. These vectors can then be used to detect information about translation, rotation and scaling. A major drawback, however, is that using SIFT in real

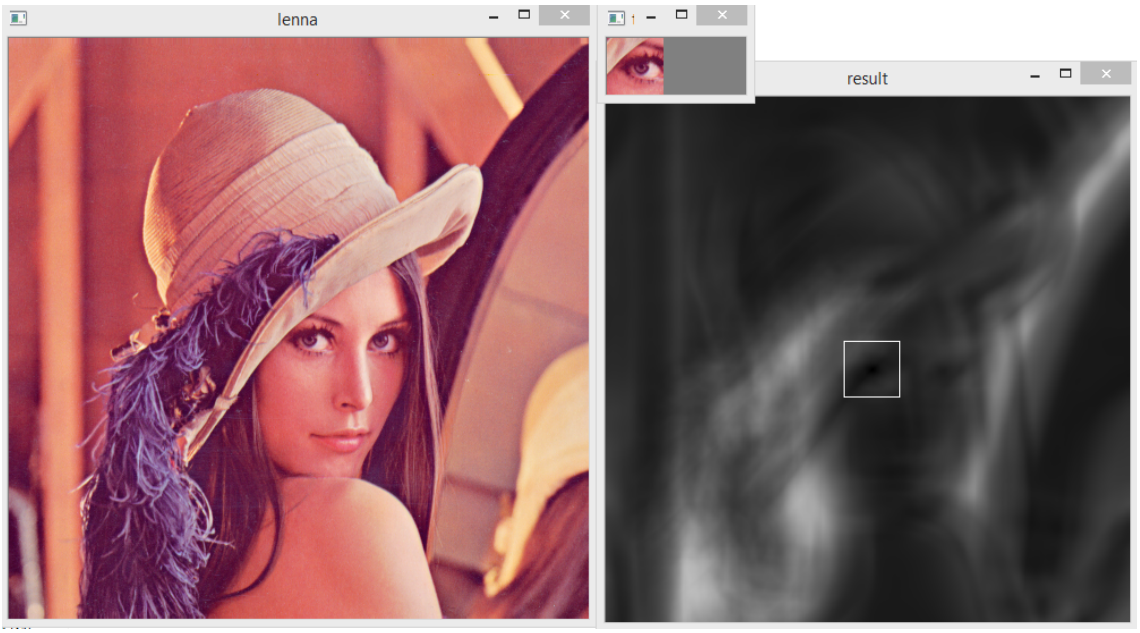

Figure 4.6: Example of template matching. The original image is the leftmost, the image in the top is the template to be matched, and the black and white image on the right is the result of the template matching. The minimum location (the best match) is surrounded by a white rectangle.

time is computationally difficult for tracking tasks, and even more so on mobile devices. Some real time implementations using SIFT have been experimented with. Such a solution was developed by Radkowski and Oliver [30], who presents a SIFT-based feature matcher for recognizing specific circuit boards. That system was run on a PC however, and its performance was not discussed in much detail. A hardware based SIFT system could also be a solution for creating a real time feature matcher. A system built by Huang et al. [31] could achieve 33 milliseconds per frame (at 640 x 480 pixels resolution) with as many as 890 features for the whole matching process.

ORB

ORB(Oriented FAST and Rotated BRIEF) [28] is a method that aims at creating a real-time runnable version of SIFT. ORB combines different methods for finding keypoints and

extracting their descriptors: FAST is used as the feature finder, and BRIEF (Binary Robust Independent Elementary Features) [32] for extracting the descriptors. The system adds an orientation for the found FAST features by calculating a center of intensity around each feature. Even though the performance differences between ORB and SIFT are quite large, the performance of ORB is still not enough for real time tracking applications, especially on mobile devices. Both of the aforementioned matchers are best used in scenarios where, for example, we have to search for an object in an image.

4.2.3 Optical flow

Optical flow (used in Section 6.3) techniques estimate the motion of pixels between two successive frames. The idea is to use brightness variations in the image to estimate the movement between two frames. There are multiple different methods that can be used for calculating the optical flow. The optical flow algorithm used in Section 6.3 is a pyramidic version of Lucas and Kanades optical flow algorithm [33]. Lucas and Kanades method assumes a small amount of movement that is constant within a certain area. The improved method proposed by Bouquet [34] further improves the calculation by using a pyramidic model, in which the optical flow is calculated for multiple different search area sizes. In order to reduce processing requirements of the optical flow algorithm, a sparse optical flow can be used (i.e. only a limited set of features are tracked instead of the whole image). There are performance-accuracy tradeoffs in calculating the optical flow as well. Using more features to track can have a positive effect on the accuracy (if the tracked features are selected well enough), but the processing power requirements can quickly become a performance bottleneck.

4.3 Simultaneous localization and mapping

One of the most promising universal visual tracking solution is SLAM, which has a large number of different implementations. The idea of SLAM is to estimate the user position in 6DoF by estimating the 3D positions of different points that are visible in the camera. The main principle behind SLAM is somewhat similar to the tracker presented in the Section 6.4. The tracker consists of multiple different parts: extraction of features, estimating the pose, and updating the system. Currently SLAM systems are used especially in robotic applications to keep track of the position of a robot.

For each iteration of the tracker, the tracker extracts a number of features from the device's surroundings. These features are then matched with the current 3D map created by the system to estimate the pose of the device. After the pose of the device is estimated, the map is extended with the features that were not previously in the map.

The difficulty in SLAM systems is to define the depth of the extracted feature points. The x and y coordinates of a feature can be easily extracted from the image, but estimating the depth requires some more complicated techniques. The most common way with monocular cameras is to estimate the depth by using the translation of the device while tracking the feature. The depth can also be estimated by using stereo cameras or depth cameras, but unfortunately such devices are not currently integrated to any consumer mobile devices.

Especially for outdoor tracking, different adaptations of SLAM have a lot of potential, since a well-implemented SLAM can track the user's position relatively drift-free in 6DoF. The downside with SLAM is the computational complexity and hardware requirements. The maps built with SLAM systems can grow very large, which can consume a lot of memory. Reusing maps can also be difficult, since especially outdoor environment might change so the map can become unusable, and it is difficult to determine automatically which parts of the map need to be updated, thus it is often easier to restart the

building of the map from a scratch whenever the system is restarted. An example of an open source SLAM system is PTAM [35]. Even though it is one of the best-known SLAM systems, it is still not suitable for larger scenes such as outdoor usage due to the sizes of outdoor maps. Implementations of a SLAM system were also considered to be a part of this thesis, but in the end they were considered too complicated to be implemented in the given timespan.

4.4 Visual 3D tracking

Lately visual trackers have also become capable of tracking 3D objects using combinations of different techniques. For example, it is possible to use 3D models as markers, where different edges and textures can be used to estimate the pose of the 3D object. SLAM (or similar) systems can also be used to create a map from an object, which can be used as a marker. As with SLAM, 3D object trackers can be improved by the use of depth or stereo cameras.

3D models can be tracked in multiple ways. The most common methods consists of searching edges in the model and edges in the image, and matching those two sets of edges to estimate the initial pose. After the pose is initialized, the tracking can be realized using any tracking technique such as using natural features found in the image or tracking different planes or textures of the object.

One advantage of tracking 3D objects is that the tracked objects can be large, previously known objects such as buildings. If the used 3D model is complete, tracking is possible from all directions. Using accurate 3D models also allows for very accurate tracking. Major downsides are that constructing accurate 3D models can often be time consuming, and tracking complex objects requires a lot of processing power from the hardware. The construction of 3D models can be avoided to, at least some degree, by using, for example, some implementation of SLAM to create a ”map” of the object which is then used for the tracking. However, using these built maps is often less accurate than tracking a complete

model. 3D model tracking has similar problems to basic marker tracking, since, for example, changed lightning conditions and shadows can create ”edges” that can easily confuse the tracker. This makes the use of 3D object tracking very difficult in outdoors scenarios. 3D object tracking also becomes problematic if the tracked target is prone to change (e.g. it is a building that is currently under construction, which would require frequent updates to the 3D model).

The most obvious industrial use for 3D object tracking is the tracking of small gadgets such as engine parts or tools. As presented in Section 2.2.1, the 3D object tracking could be used to view for example maintenance instructions. For larger scale applications, tracking a building can be considered as a solution for outdoors tracking. However, the issues with varying conditions and shadows can be difficult to overcome, and, therefore, it is not often worth to adapt 3D tracking to larger scenarios. It should be noted that as depth camera technology improves, 3D object tracking will become a lot more useful in outdoor use. The current ranges of depth cameras is still less than 10 meters, which makes them unsuitable for outdoors use.

4.5 Combining different tracking types

Non-visual tracking is mostly used for outdoors applications, when no prior visual knowledge of the area exists. Non-visual tracking applications depend on sensory data of the devices. These sensors usually consist of a Global Positioning System (GPS) sensor, magnetometer, gyroscope and accelerometer. The accuracy of non-visual tracking is most of the time smaller than visual tracking, since the accuracies of the different sensor can vary a lot. Sensors are discussed in more detail in Chapter 5.

As mentioned by Azuma et al. [15], combining several different tracking methods is the most reliable method for making reliable tracking outdoors. By combining different solutions, it is possible to fix a large number of major problems that arise from using just one tracking system. This thesis proposes a system which combines sensory data

from gyroscope, GPS and magnetometer with a visual tracker. Even though the article of Azuma et al. is relatively old, the basic idea is still valid. Pure visual tracking should not be the only form of tracking, since the environment can vary quickly when the lightning conditions change, and visual trackers can be lost during fast movements or if the camera view is blocked. When visual tracking fails, *Inertial Measurement Units* (IMUs) can be used to restore the devices orientation, and IMUs can also be used to support the visual tracker by comparing the values of the tracker and sensors. It is easy to determine that there is something wrong with the visual tracker, if suddenly the orientation given by IMUs is significantly different from the visual trackers orientation. The visual tracker can be used to compensate for the drift of the sensors, assuming that the used visual tracker is drift-free.

A combination of different visual tracking systems is also possible. Nowadays many AR SDKs provide a tracking method known as extended tracking. In extended tracking, the user initializes the pose estimation by pointing the camera to a known marker. After the initial orientation is computed, the tracker starts to build a map of the environment around the marker by some technique such as a variant of SLAM. This allows the user to point the camera away from the original marker, but the tracker is still able to estimate the position of the user with the map. Whenever the marker becomes visible again, the tracker can correct the orientation of the device if any drift error has been accumulated.

The best results can be achieved by using a drift-free visual tracker for the main tracking, while the system can be supported with different sensory data. A complete outdoor tracking system could consist of, for example, a sensor module which uses a combination of gyroscope and compass to calculate the absolute orientation of the device, a GPS to find out the absolute position of the device, and a drift-free visual tracking to do the main tracking (that can also be supported by different IMUs).

Chapter 5

AR Technology

A large number of different tools exist for creating AR applications. This chapter discusses both hardware and software that can be used for AR development. The hardware section focuses mainly on different sensors that can be found in modern mobile devices. The software section focuses on different software development kits and libraries that can be used for creating AR applications.

5.1 Hardware

AR applications can often have very demanding hardware requirements. The used devices must have a lot of processing power, since computer vision algorithms required for pose estimation can be computationally difficult. IMUs are also a common requirement for AR applications since they can be used for different pose estimation tasks, such as using GPS and magnetometer for determining absolute position and orientation in the world coordinates or using gyroscope to supplement visual tracking.

5.1.1 Mobile devices

The rapid development of mobile technology has allowed AR to reach the hands of regular customers. Previously AR could only be used in PCs or dedicated tools such as AR

glasses/helmets, which are often very expensive. As mobile technology has developed, basic mobile devices have become capable of running augmented reality applications. Especially tablet computers can nowadays run AR applications that require a lot of processing power. Most of the modern mobile devices today also include a wide variety of sensors as well as high quality cameras, which AR applications can utilize.

5.1.2 Sensors

Different sensory data can be used to track the movements of the user fairly accurately. The most important sensors for outdoor tracking are GPS, gyroscope, accelerometer and magnetometer. These sensors are mostly used since they are often included in regular mobile devices.

Gyroscopes can be used to approximate the rotations of the device. The accuracy of gyroscopes can be quite high, since the gyroscopes are not affected by any outside disturbances. The gyroscopes can also track fast rotational movements with good accuracy. However, the gyroscopes tend to accumulate a drift after they have been used for a while. Also, the gyroscopes do not have any absolute positioning which could be used to alleviate the drift. For example, a Nexus S phone gyroscope test conducted by Subbu and Dantu [36] showed that when rotating the device around a single axis for 90 or -90 degrees, the error remained below 6 percent. However this can become an issue after a large amount of rotations have been done, since the drift usually keeps on accumulating in a linear fashion. If the purpose of an AR application is, for example, to view some wiring inside a wall, the drift error is already much too severe. A compensation for the drift is required, and one example of such compensation is explained in the Section 6.3, where another tracker is used to reset the values of the gyroscope.

Accelerometers are used to measure the acceleration of the device. Where the gyroscope measures the rotation of the device around the three axes, accelerometer is used to measure the translation. Modern mobile devices are often equipped by three accelerome-

ters, one for each axis. By combining gyroscope and accelerometer, a device can estimate a full 6DoF pose. However, accelerometers also suffer from the accumulation of drift, which means that if a tracker is based solely on the inertial measurement units of the device, some method for drift compensation is required. These methods can include, for example, the use of magnetometers and visual trackers.

In order to determine the absolute orientation of the device, magnetometers can be used. One issue with magnetometers is that the accuracy is highly affected by outside disturbances, such as large metal bodies or electric wirings. Raw magnetometer data is also often very noisy. The noise of the magnetometer can be reduced by for example using data from a gyroscope for filtering, as is done in sensor fusion system in Section 6.2. A bigger problem for magnetometers is the sensitivity to outside disturbances. Those disturbances are often impossible to predict and compensate, which makes the use of magnetometer unreliable when a high accuracy is required.

The most commonly used outdoor positioning system is GPS, which can be used for tracking the position of the user almost anywhere in the globe. However, the accuracy of the GPS on basic mobile devices is around 5 meters. Multiple technologies exist to improve the accuracy of GPS. For example, Fong et al. [37] present a GPS solution which can reach accuracies of 10 cm.

For positioning, GPS is not the only method. If the coordinate system does not need be global (i.e a relative coordinate system can be used), some smaller scale positioning systems can be used. For example, ultra wide band or bluetooth positioning systems can provide a level of accuracy that is sufficient even for industrial use cases. However, most of these systems require some extra work, such as placement of beacons in specific locations.

The quality of visual tracking is highly dependant on the quality of the camera. Usually video resolutions upwards of 640 x 480 are enough for tracking purposes, but the resolution is not the only thing that matters. Low quality cameras are often heavily dis-

torted, and have very low focal lengths, which can cause major problems in tracking. The images often become curved on the edges, which can cause methods such as edge detection to work poorly on the edges of the image. Distortions also need to be compensated when building a visual tracking system. Pinhole cameras are also affected by vignetting, where the pixels closer to the edges of the image are darker than the pixels in the middle of the image. Vignetting can also be compensated by, for example, using a diffusely lit white target visible in the image to calculate the amount of vignetting in different parts of the image [17]. Other compensation methods also exist for taking into account other distortions. The OpenCV library offers a toolset for calibrating the camera by taking images of a chess board pattern. The distortion can be estimated from the curvatures of the assumed straight lines on the board. When the parameters of the camera are known, distortion can then be removed. These distortions can be quite problematic when trying to build a solution for customers, since the amount of camera types that are in use in different devices is very large, and all of them would have to be calibrated separately.

For visual tracking, different cameras are also available in addition to the basic monocular cameras. Both stereo cameras and depth cameras can be used to help extraction of three-dimensional data from features. Where a monocular camera requires horizontal movement of the device to determine depth of the features, stereo and depth cameras do not. Stereo and depth cameras can be useful in tracking of 3D objects, or in SLAM system implementations. However, one issue with these cameras is that they can not be found in any consumer grade mobile devices, so integrating them into a tablet or mobile phone would be difficult. In the near future, depth sensors might become commonplace in mobile devices as well. Some standalone depth cameras (such as Structure Sensor [38]) already exist, and, for example, Dell is including Intel's RealSense depth sensors in some of their new tablets (Dell Venue 8 7000 Series) [39]. The problem with depth sensors is that at least in their current state, the usefulness of depth sensors outdoors is very limited, since they usually have a sub-10 meter range. Stereo cameras can have a larger range for

sensing depth by separating the cameras further apart from each other.

5.1.2.1 Sensors in mobile devices

Most of current mobile devices (mainly tablets and smartphones) have at least a camera, GPS and some sort of compass. Majority of these sensors are accurate enough, and can be used in tracking systems. However, there can be some differences in the quality of the sensors. One of the most noticeable differences is the GPS quality of different mobile devices. Both the accuracy and update frequency of the GPS can vary. If the device is not the cheapest available, accelerometers and gyroscopes are also included most of the time. Some sensors can also be connected to the mobile devices, such as the Structure Sensor depth camera mentioned in the previous chapter.

5.1.2.2 Standalone sensors

Some external sensor modules are also available for mobile devices. Examples of such sensors are Texas Instruments Sensor Tag [40], and NODE+ [41] device. Sensor Tag and NODE+ can both be connected to a mobile device, in the case when some required sensors are missing from the device itself. The accuracy from external sensor modules can be better than the accuracy of the inbuilt sensors. Since some mobile devices might use lower-end sensors, and external sensors might also include more effective filtering or even sensor fusion functionality, it could be beneficial to use these external sensors as they might provide higher accuracy.

Especially in the case of GPS, sensor accuracy can be improved much by using professional grade external GPS receivers. Some higher end GPS receivers claim to be able to reach sub-1 meter accuracies. External sensors are often difficult to connect to tablets for example, which lowers the usability. External sensors can also often require placement of beacons in specific spots, which can cause inaccuracies if positioned wrong. Beacons are usually small devices, and a varying amount of beacons need to be placed for different

indoor positioning systems.

5.1.3 AR devices

Since it has been shown that AR technologies have large potential in a number of different fields [42], there have been large advances in developing devices that focus solely on AR applications. These devices have large variances: some of the devices are expensive developer-only devices, and some are much cheaper consumer products. However, they still have large problems that are discussed in this section. The presented problems are also the reason why they are not used in this thesis.

AR (and VR) devices have two major challenges: performance and displays. It is difficult to get a large amount of processing power into standalone AR glasses (such as Epson BT-200 [43]). Glasses which can be used as another screen for PC's (such as Vuzix M2000AR [44]), and virtual reality helmets like Oculus Rift [45] can be difficult to use in outdoor environments since a PC is required. The displays of AR glasses are usually quite small and they have a small field of view (FoV), which makes them unsuitable for professional AR tools. The visibility of the translucent screen of AR devices outdoors is often very poor. In their current state, AR devices are only usable for viewing, for example, some text, since augmenting any large 3D models is difficult due to the limited field of view, which would mean that only a small part of the model can be visible at a time.

Some non-commercial AR tools can also be found from literature. For example, the systems presented by Menozzi et al. [16] and Schall et al. [18] are both custom solutions. By building the systems from a scratch, the systems can be optimized to a further degree which can allow for more complicated tracking algorithms. Both systems included some inertial measurement units such as GPS, magnetometers and gyroscopes. However, one could argue that these types of solutions are becoming more and more obsolete, since modern handheld devices already include a large number of sensors and high quality

cameras, and even depth cameras in the future.

The social acceptance is also something that should be taken into account when considering standalone AR device, as pointed out by Carmigniani et al. [46]. Many of the standalone AR devices can be large and obtrusive, which can make people unwilling to use them. A large number of people also considers it inappropriate to wear AR devices that includes a camera which is always recording. Since mobile devices are already quite commonplace, their use in AR applications can be considered more acceptable.

5.2 Software

Since the AR technology has become more and more popular during the last years, a large number of different libraries and software development kits have been released that support the creation of AR applications. In order to select suitable tools for this thesis, a small scale research was done in order to map out some of the possible software solutions for implementing a tracking system.

There is a large variance on the quality and ease-of-use of these libraries. Some AR tools are simple content management systems that have a drag-and-drop interface that can be used to create AR applications using images or locations as targets and that can view different kinds of content such as images, videos and 3D models on the target. A large amount of lower-level development tools also exist, which only provide a tracking functionality but require some amount of programming in order to create the actual applications.

5.2.1 Suitable SDKs and libraries

A large number of smaller libraries exist that support the creation of sensor-based AR applications for outdoors use such as DroidAR [47], PanicAR [48], 3DAR [49] and BeyondAR [50]. However, these smaller libraries are mainly meant for creating navigation-

and other applications which do not require a high quality tracking as different visualization applications do. These smaller libraries are also sometimes relatively difficult to use and outdated, since they are not always updated frequently.

One of the largest AR SDK providers today is Wikitude [7]. Their SDK supports creation of multiple types of AR applications, including sensor-based AR. The Wikitude SDK also provides possibility for using image/template based tracking systems. Although Wikitude's SDK is relatively easy to use, the SDK also has its flaws. Since the SDK is a commercial product, large parts of actual functionality is hidden. This means that adjusting the tracking is currently very difficult. The SDK does not do any visual tracking to support the geolocation-based AR. Therefore, applications have to rely heavily on the use of compass. In order to create very accurate tracking systems, it is not recommended to use magnetometer data for determining the devices orientation. Wikitude's SDK is also going to have a SLAM tracking system in the near future, as the current version of the SLAM tracker is on a beta stage. For the thesis, a test application for Wikitude was developed, and it is discussed in more depth in Section 6.1.

One of the largest computer vision library available today is the open source library OpenCV [24]. OpenCV itself does not include any complete tracking algorithms or possibilities for 3D model rendering, since it mostly consists of low level image processing algorithms. However, OpenCV can be used in combination with different game engines such as Unity 3D, which makes it a powerful development tool. A large number of AR trackers are based on OpenCV, and all of the tracking systems developed for this thesis are based on the OpenCV library.

5.2.2 Other development tools

One of the challenges in creating AR applications from scratch with OpenCV is 3D rendering. Using an image processing library such as OpenCV does not include 3D rendering capabilities, which can cause for a large amount of work. One possible tool for supporting

the creation of AR applications is Unity3D [1]. Unity3D provides a large number of tools that can be used to create AR applications: Unity3D can handle UI creation, 3D rendering and building to Android, iOS and Windows devices. Recently, the latest iteration of Unreal Engine [51] has also shown some interest towards AR application support. Using these game engines is helpful when integrating different tracking systems together, since tracking systems can be implemented as Unity 3D plugins.

Chapter 6

Built tools

The work for the thesis has been split in to four different categories: Wikitude application, Sensor Fusion algorithm, Natural Feature Tracker and Panorama Tracker. Each of these projects are described in their own chapters. The results for each of the chapters are described in Chapter 7.

All of the applications were developed and tested on a Lenovo T440p laptop [52]. The tablet that was used for testing is Nvidia Shield [53]. On PC version, Logitech C920 webcamera [54] was used.

All of the used applications (excluding the commercial Wikitude) were targeted for Unity 3D, since Unity 3D allows easy development for multiple platforms, and Unity 3D helps the integration of different systems together. The applications developed in the thesis were run on PC as well as on Android device which has a gyroscope and a magnetometer. All image processing is done using OpenCV library.

6.1 Wikitude demo application

In the first phases of the thesis, commercial solutions for building AR applications were reviewed, and Wikitude SDK was selected as the most promising solution, since it is one of the largest AR SDK providers today and it also has a free trial version that can be used

for testing. Wikitude SDK provides a number of tools for creating geolocation and image tracking based AR applications, and it is also relatively easy to use. Therefore, a demo application was fast to build with Wikitude SDK. The idea was to test the viability of commercial solutions in building industrial-usable AR applications that could be mainly used for different visualization tasks, for example, viewing a 3D model of a building in its correct position before it is built. The aim of the demo application was to find out the accuracy and potential limitations of pure IMU tracking.

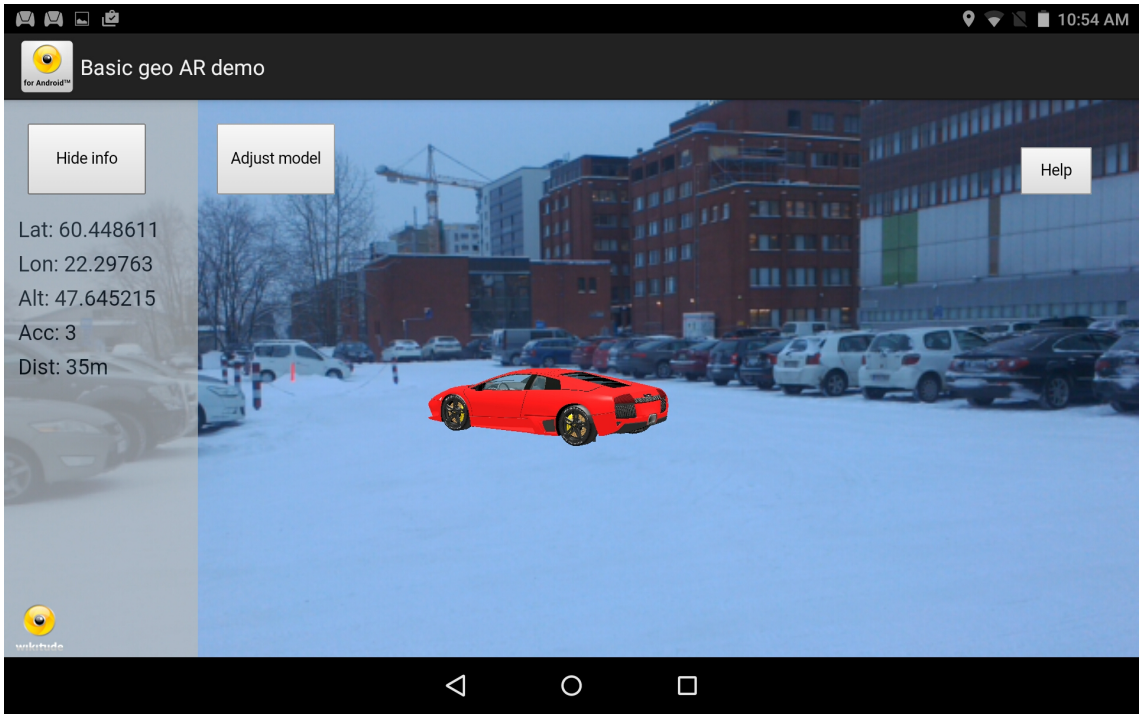

Figure 6.1: Image from the Wikitude demo application

6.1.1 System functionality

The basic idea of the application was to show some augmented content such as buildings, in outdoor environment. The plan was to build an application which could be used as a visualization tool for construction. Wikitude tracking relies solely on the GPS and IMUs of the mobile device. The exact functionality of the SDK is hidden, which means that it is unknown how different sensors are used.

Wikitude application uses GPS to determine the users position, and 3D models can be augmented to a pre-defined GPS coordinates. Two different use modes were implemented mainly to find limitations regarding the inaccuracies of the GPS system.

1. The basic use mode was a plain AR-browser like application, in which the application determines the location of the user according to the GPS, and the orientation of the device was determined by different IMUs of the device.

2. The second mode was implemented to account the inaccuracies in the GPS. This mode used a simulated GPS, where the location of the user was defined beforehand and the GPS was pinned to that location. The pinned location allowed testing the accuracy of the tracking using the sensors when the inaccuracies of the GPS were not taken into account, since sometimes the GPS can cause large jumps in the augmented content when the position is updated. The results for testing Wikitude are presented in Section 7.1.

6.2 Sensor fusion in Unity

After finding out the limitations with Wikitude SDK, other tools were studied. The next step was to start building a custom tracking solution from scratch. Firstly, the noisy magnetometer data needed accuracy improvements in order to have a system that allows the estimation of the device's absolute orientation. The noise removal was accomplished by combining the sensor data from the gyroscope with the data from the magnetometer. This fusion also aims at making the augmentation look more appealing to the user, since the augmented content would not jitter so much because of the noise.

6.2.1 System functionality

The fusion of gyroscope and magnetometer was done in a fairly straightforward way: the orientation was determined from both the gyroscope readings and the magnetometer

readings, and they were combined every frame using a defined coefficient value (a value between 0 and 1) which determined how much each of them affected the result. For each axis, the combined orientation was calculated as follows:

$$\text{orientation} = (\text{gyroReading} * \text{coefficient}) + (\text{magnetometerReading} * (1 - \text{coefficient}))$$

After testing, the used coefficient value was set at 0.9. The coefficient value should be set so that the gyroscope's orientation effects the result more than the magnetometer's orientation to remove as much noise as possible. Since the magnetometer does not accumulate any drift, the drift of gyroscope could be accounted for by reseting the orientation of the gyroscope each frame and by setting magnetometer's orientation to the gyroscope's orientation. In the final Unity 3D version of the application, previously defined augmented content could be placed on the screen according to the orientation from the formula presented earlier. This simple augmentation was implemented to visualize the difference between the raw magnetometer data, raw gyroscope data and the combined data. The results of the sensor fusion algorithm are presented in Section 7.2.

6.3 Natural feature tracking

Due to the disturbances of magnetometers, it is desirable to depend on magnetometers as little as possible. In order to make that happen, a visual tracking was combined with the sensor fusion system. The idea of the visual tracking is to supplement the magnetometer and gyroscope fusion so that the magnetometer is only required in the beginning of the tracking to determine the absolute orientation, or when the visual tracker accumulates too much drift. If the magnetometer is only used for getting the initial orientation, a visual tracking system which minimizes the drift accumulation is required. The first iteration of the visual tracking was built with OpenCV, and the visual tracker calculated a sparse optical flow for a set of points in the image. The idea of the sparse optical flow is to estimate the movement of a certain set of points, rather than the movement of every pixel

of the image. This tracker was built to test the performance of a simple natural feature tracking system and see how much drift is accumulated and in which types of scenarios the method could be used in. The visual tracker only computes a 3DoF pose, since the depth of the tracked features is not taken into account.

The tracking system was ported to two different plugins in Unity 3D: one for debugging in the editor on PC, and another for running in Android devices. The tracker was originally developed as a C++ application on PC.

6.3.1 System functionality

The Unity 3D application consists of four different usage modes: compass-only, gyroscope-only, natural feature tracking (NFT) only or combined mode, in which the natural feature tracking is combined with the sensor fusion presented in Section 6.2. Changing between different usage modes was required in order to make the testing and comparing of different systems easy. A simple augmentation with a cube object was also possible to accomplish, since it was required to test, if the initialization using magnetometer was functional and to visualize the drift and robustness of the tracking. Due to the simplicity of the system, the tracker was only capable of tracking the rotation of the device but not translation. The initialization was required to be run again whenever the user moved to a different position. The system did not include any absolute positioning system such as GPS, since it was only supposed to be used for the testing of the tracking accuracy. Adding a GPS support would have been a simple task, but due to the accuracy issues of plain GPS the GPS support was not implemented.

6.3.1.1 Initialization and tracking

Whenever the user pushes the ”Start Tracking” button in the UI, the previously defined AR content is placed to its correct position according to the orientation of the device. The initial orientation is computed by the sensor fusion algorithm described in the previous

chapter. After the content is positioned, the NFT is initialized by searching the whole frame for trackable features with the method proposed by Shi and Tomasi [23] (explained more accurately in Section 4.2.1). For the version run on tablets, maximum of 50 features were searched for with a minimum quality of 0.5 and minimum distance of distinct features 25 pixels. All the found features were added to a vector of points. However, often the number of tracked features were far below the minimum, since no good enough features were found.

After the features were found, tracking was started. For each feature, the optical flow was calculated using pyramidical version of Lucas and Kanade's optical flow algorithm [34] (see Chapter 4). The amount of movement was calculated by taking the movement of each feature along y-axis and x-axis and by calculating their averages. Estimating the rotation around z-axis was not considered important, since the testing was more focused in getting a general idea of how accurate the tracking is, and it was assumed that the user would keep the device straight all the time. In order to compensate loss of features that are dropped when they move out of screen, new features were searched whenever the estimated movement of the camera exceeded 100 pixels (with the camera resolution being 640x480 pixels) in either horizontal or vertical direction. Since these updates were only run on a small part of the frame, they did not have a noticeable effect on performance.

6.3.1.2 Compensating for the drift

Since the system does not have any sort of reinitialization system or a system which would locate points that have been previously tracked, the system accumulates drift over time whenever the user moves the device in wide angles, which causes the system to search for new points frequently. In these cases it is required to recalibrate the augmented content back to the correct position as defined by the sensor fusion system. In the final system, the recalibration happened in static intervals of 5 seconds, or manually when the user presses a reset button. The results of the tests for natural feature tracker are presented

in Section 7.3.

6.4 Panorama tracker

The most time consuming part of the thesis was the development of the panorama tracking system, which aimed to correct the problems (mainly the drift) present on the simple natural feature tracker. The main reason to start developing the panorama tracker was the fact that a tracker which accumulates no drift would remove the need of compass altogether, except for the initial calibration. It is also possible to calibrate the absolute position and orientation by using some other methods besides compass, such as different visual recognition techniques. The main functionality of the panorama tracker is based on the article by Wagner et al. [17]. The basic idea of the tracker is the same as in the paper, but some parts of it have been changed. The largest difference is the integration of the tracker to Unity 3D, which also allows the augmentation of virtual content. The developed system also has some different methods for calculating the orientation from the tracked features. The system presented in the paper uses iterative Gauss-Newton method for estimating the orientation more accurately, but such method was not implemented during the thesis. The system presented in the paper maps the image to a cylinder, in which each point is presented in 3D form. The system developed in the thesis stores the whole map in a single 2D image. Multiple smaller differences are also present.

6.4.1 System functionality

The panorama tracker creates panoramic images of the surrounding area, which are used to track the rotational movements of the camera. The maps are created parallel to the tracking, so the tracker does not require previously built panorama images. The main problem with this type of tracking system is that it does not work with translational movement; whenever the user moves to a different position, a new panorama map has to be

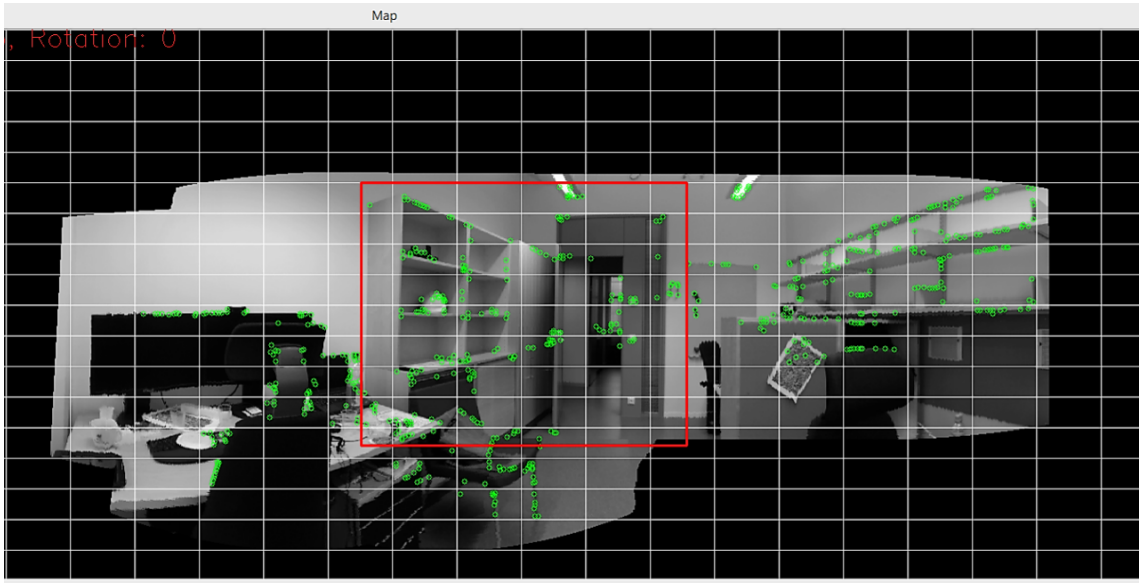

Figure 6.2: Partial panorama on PC. The current viewpoint is visualized as the red rectangle, cells as white rectangles and tracked features as green circles.

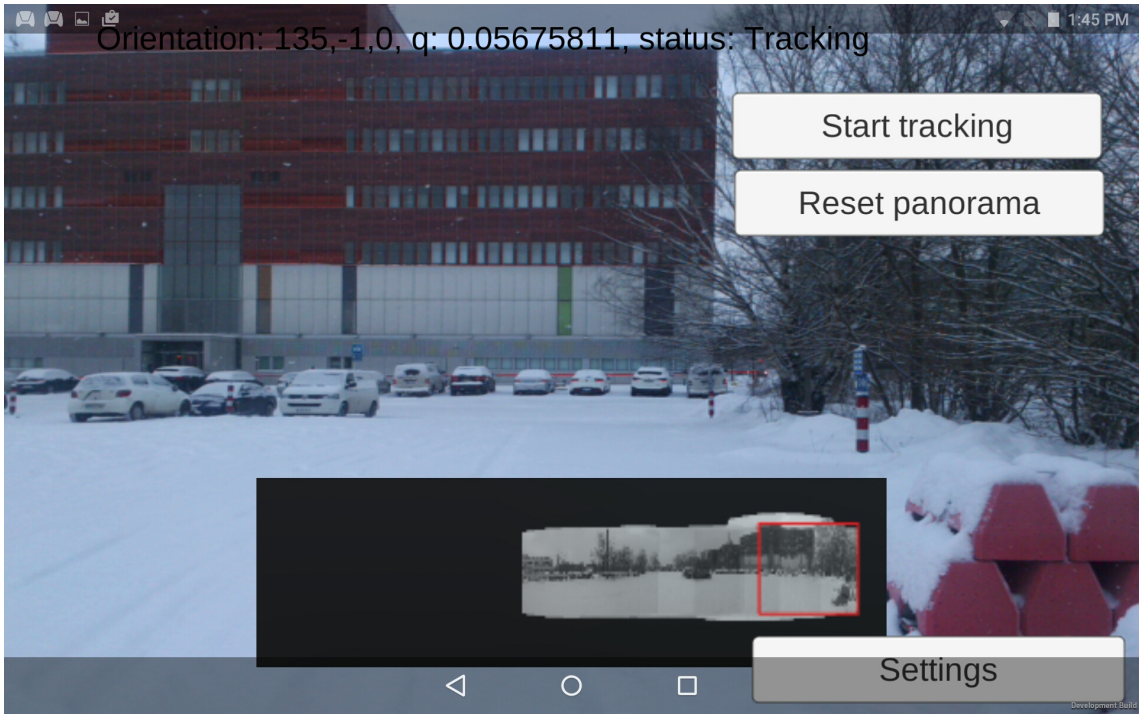

Figure 6.3: Image from the UI of the panorama tracker on Android Unity 3D

created. With mobile devices the tracker can also be configured to use the gyroscope to support the tracking. For example, when the visual tracking is lost and the relocalizer

is started, the tracker can visualize the augmented content using the gyroscope until the tracker has reinitialized. It is also possible to combine the orientations of the gyroscope and the panorama tracker in a similar fashion as the sensor fusion (presented in Section 6.2) combines the orientations of the gyroscope and magnetometer. Especially, when the tracking quality is detected to be low, using the gyroscope can be more accurate.

The tracking is done using a set of features that are template matched to the current images from the camera, and the map is updated each frame whenever the camera covers new area that has not yet been added in to the map. The system also includes a relocalization system, which is based on the template matching. A certain similarity to SLAM systems can be seen, since the basic idea of the panorama tracker is very similar: each frame the estimated orientation is updated, and new features are added to the map. The flow of the system is explained in Figure 6.4.

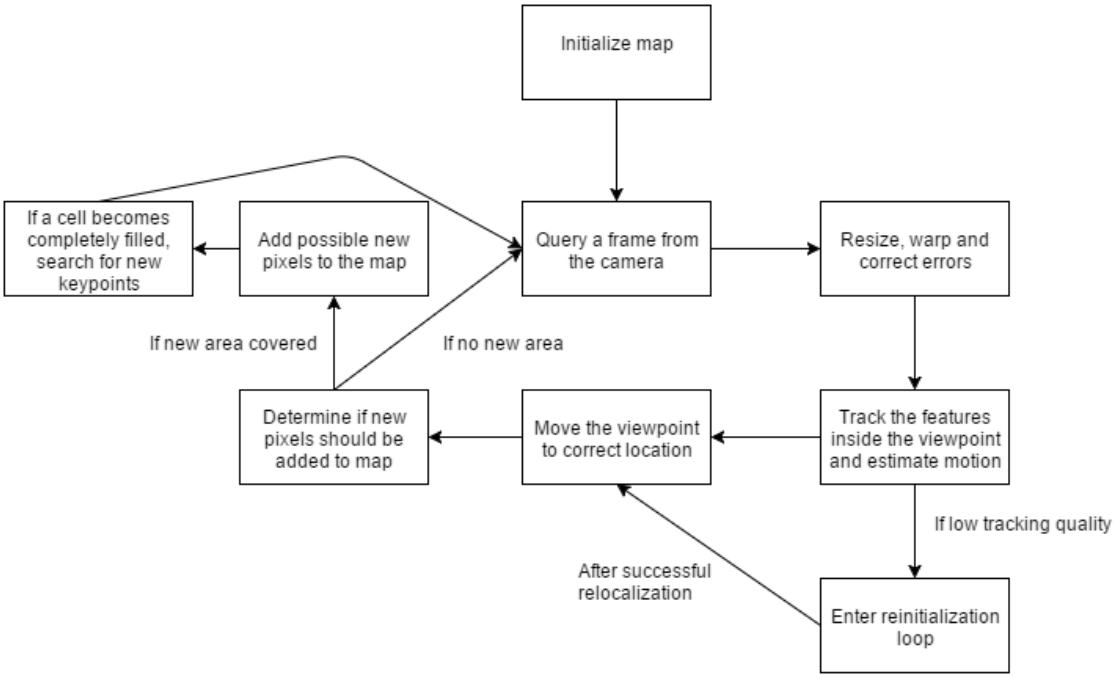

Figure 6.4: Control flow inside the application.

6.4.1.1 Camera calibration

The first step in the initialization of the panorama tracker is image correction. The current system does four different operations for each image: grayscale conversion, camera distortion correction, warp the image based on the current rotation of the camera and scale the original frame to a half- and a quarter sized.

The grayscale conversion is done in order to improve the performance of the system. Template matching is already quite slow, and using color images makes it even slower. Using colored images has no advantage in the quality of the tracking either, so using grayscale were used. Colored images were tested with the template matching, but they provided no difference in the quality of the tracking. The only difference between colored- and grayscale images is that the performance is a lot lower when using colored images.

The camera distortion is fixed using the camera calibration functionality that is found from OpenCV. The calibration is done by viewing a chessboard pattern from different angles, after which the OpenCV calculates the intrinsic camera parameters that can be used to fix the distortion that usually appears on the edges of the image. Each image is also warped based on the rotations around z-axis and y-axis. The warping is done so that the images from the camera can be mapped to a (cylindrical) panorama image. Without warping the images, creation of the panorama image would be impossible. The image warper in OpenCV is also used to resize the images. A sample of the warper functionality is presented in the Figure 6.5.

Attention should also be paid to the settings of the camera. Especially auto-focus, auto-gain and auto-exposure can become an issue. Disabling the automatic parameter adjustments can improve the tracker quality, but in some cases disabling the automatic adjustments is not possible. The issue with automatic parameter adjusting is especially problematic due to the usage of template matching as a tracking method for features. Since template matching is a simple procedure, it does not take changes in brightness into account very well. In outdoor scenarios the removal of these automatic parameters can

Figure 6.5: Functionality of the warper: the original image on the right, and the resized, grayscale warped image on the left. The image is warped to represent the change when the camera is rotated 10 degrees on the y-axis.

also become a problem in some cases, since the image might become way too saturated due to varying lightning conditions. In the tablet version of the tracker, these camera settings were left untouched, since adjusting them using Unity 3D was not possible. Motion blur is also problematic when the camera is rotated quickly, since the system has no implementation for determining whether or not the current image is blurred. This can cause the system to add blurred data to the map, which is often difficult or impossible to track.

Since the conversion from pixels to degrees (explained in more detail in next section) is dependent on the field of view of the camera, a method was required to estimate the FoV as accurately as possible. For a web camera that was used with the laptop, the FoV information was available online, but for the tablets this data was not found. The FoV was estimated by placing two images on a wall, and the distance between the images was measured. The camera or tablet was then placed on a tripod, and moved to a position where both of the images were on the edges of the camera image. Then the distance of the camera was measured from the wall, and the field of view could be calculated.

6.4.1.2 Panoramic maps

The system creates three maps, which are used to create a pyramidical method for tracking. The maps cover 360 degrees in horizontal direction, and 60 degrees in vertical direction. Limiting the vertical movement reduces the map sizes, and most of the time it is not required to track very far on the vertical direction. For the visualization tasks that the tracker was built for, larger vertical movements are also not necessary, since beyond the 60 degrees nothing else is often visible than the sky and the ground, which are difficult to track accurately.

The size of the map is dependent on the FoV, the resolution of the camera and the covered degrees of the map. For example, on PC with image resolution of 640x480 pixels, horizontal FoV of 55 degrees and vertical FoV of 43 degrees the resolution of the map was 2940x655 pixels. The map was set to cover 420 degrees of horizontal movement. The application does allow to change the map size in the settings menu by scaling the images from the camera, if performance becomes an issue or if the tracking quality needs to be higher and the hardware is capable. The smaller maps are a half- and a quarter sized versions of the biggest map. Limiting the map sizes has a direct effect on the performance of the tracking, but it also lowers the tracking quality. The maps are also split into 32x18 equally sized cells. These cells are used for two purposes: updating the smaller versions of the map, and for searching new features. Updating the map is explained in Section 6.4.1.4, and the searching of new features in Section 6.4.1.3.

The orientation of the device is stored to a viewpoint rectangle. The rotations around x-axis and y-axis are saved as pixel positions in the map, which can be translated to degrees when the resolution and FoV of the camera are known. The rotations around x-axis and y-axis are calculated using the following formula:

$$\text{degrees} = (\text{pixel position}/\text{map width or height}) * 360$$

The information about rotation around z-axis is not stored in the viewpoint, and it is only

taken into account when warping the image as presented in the previous chapter. The rotation around z-axis is stored as degrees instead of pixels.

6.4.1.3 Tracking

The actual tracking detects features that are searched with a FAST (described in Section 4.2.1) feature search. FAST detector was selected due to its good performance, and since the points selected by FAST rarely have points that are difficult to track. Some other detectors were also tested, such as ORB, Harris corner detector and the method proposed by Shi and Tomasi [23], but the quality differences in the features were not very high, and other detectors provided no advantage over FAST. Performance of the feature detector was not important, since the features are not searched every frame.

The feature search is based on the cells within the map. Each of these cells contain their own list of features. The feature finder is applied after a cell gets completely filled with pixels in the map. The maximum amount of features per cell can be adjusted to either affect the performance or the quality of the tracking. The final system used a maximum of 10 features per cell. With larger amount of features per cell the tracker often ended up selecting features with too low quality, so using more features was rarely useful. The threshold parameter used for FAST in the system was set to 15.

The tracking always assumes that the pose during the previous frame is known in order to limit the area from which the selected features are searched for. The current pose can be seen in the images as the red rectangle. The requirement for knowledge of the pose in the previous frame also means that when the tracker is initialized, a guess of the initial orientation is made. On mobile devices, IMUs could be used to estimate the initial pose. However, using IMUs for the initial pose was not implemented, so the initial orientation was always set to (0,0,0) on both PC and tablet. This can cause some issues, if the initial pose is wrong. The created map becomes highly distorted if the initial rotation around y-axis or z-axis is incorrect.

The tracking is done in three different steps, one for each map, beginning from the smallest resolution map. The current frame of the camera is given to the system, and the image is warped according to current estimated pose. Since the system has a knowledge of the orientation of the previous frame, the new orientation can be assumed to be close to the orientation of the previous frame. The current frame is first resized to quarter resolution to make it matchable with the quarter resolution map. For each cell that is estimated to be inside the current view, each feature is iterated through. For each feature, an 8x8 pixel template is extracted from the map. Then, a search area is extracted from the current frame inside which the location of the feature can be assumed to be. Since the calculation is done each frame, it can be assumed that the new location of the feature is found near to the old location. Therefore, the template matching is run on a small area. The size of the search area is different for the three maps: the quarter resolution map uses 16x16 pixels, the half sized map uses 12x12 pixels and the largest one uses 12x12 pixels. The sizes were selected by testing the tracker with different settings to find good balance between quality and performance. Larger search area sizes found more false positives. The false positives can happen, when the search area is large and might contain multiple areas that look similar, which can cause the template matcher to select the wrong location. Matching the template to a larger search area size also lowered the performance. Smaller search area sizes make the tracking a bit more robust, but if the search area sizes are dropped more the amount of movement that can be done between each frame becomes very small. Since the position estimation starts from the smallest map, and the viewpoint is moved towards the estimated position after each map, the search area can be kept small for each map.

The actual matching of the support areas in the search areas is done using template matching. Template matching is a simple method for matching similar areas, and template matching can be efficient when used in a small area. Template matching was selected since it is a fairly effective method whenever the matched templates are very small, and

after some testing template matching was found to be quite reliable when the feature actually exists in the searched area. Since the panoramic tracker only functions in 3DoF, the issue of scale invariance in template matching is ruled out. Rotation invariance can also be accounted for: as template matching can work if the rotation is very small, the rotation can be estimated by calculating the a rigid transformation between each frame. The rigid transformation is calculated with an OpenCV function estimateRigidTransform. The rigid transform estimation function takes two vectors of points as input. One of the vectors holds the tracked features from the previous frame, and the other vector holds the matched features from the current frame. The rotation can be extracted from the matrix that is provided by the function. This estimated rotation is provided for the warper, which rotates each frame accordingly. The rotated frames are then matched against the map. Out of the available methods template matching was also one of the fastest for this scenario, since the compared templates and areas are very small.

OpenCV provides multiple methods for matching the templates, but the differences in tracking quality or performance were not large. The selected template matching style (CV_SQDIFF_NORMED) in OpenCV is used to return the best match for each feature, and since it also returns a quality value, it can be used to filter out bad matches or false positives, and to estimate the overall quality of the tracking. More detailed explanation of template matching method can be found from Section 4.2.2.

The movement between the current and the previous frame is estimated quite similarly to the NFT, with a few differences. The x- and y-movements are calculated for each frame, and their averages are calculated. A simple filtering was also implemented to remove the obviously faulty matches. The filtering module calculates the median for both x- and y-translations, and removes points which differ more than a threshold. The maximum difference between the median and the point was set at 2 pixels.

6.4.1.4 Updating the map

Whenever the quality of the tracking (given by template matching) is sufficient, the map is updated. The update is computed once per frame, whenever there are new pixels inside the camera view that are not yet inserted to the map.

The updating of the map is based on three different binary mask images: the mask of currently mapped areas in this frame and in the previous frame, and the mask of the current frame. Mask of the current frame is required since the warping of the images causes a number of black pixels on the edges of the image, as can be seen in Figure 6.5. The first step in the updating of the map is to calculate a bitwise-or between the mask of the current frame, and the mask of the whole map in order to extract the coordinates of the pixels that can now be added to the map. The resulting pixels are then extracted from the frame and set to the map. After setting the pixels to the map, the corresponding pixel positions are added to the mask of the whole map.

The smaller maps are not necessarily updated every frame, since copying and resizing images takes a large amount of processing. Because of the computational cost of resizing images the smaller maps are only updated whenever a cell in the map becomes completely filled. Whenever that happens, the pixels in the filled cell are extracted and resized, and placed in their correct position in the smaller versions of the map. New features are also searched for when smaller maps are updated.

6.4.1.5 Saving and loading maps

The maps can be saved and loaded on the PC version. The process of storing the maps is straightforward: the application stores text files which contain all the information about every keypoint and which cell they belong to. All three maps and all the masks are saved as plain png images. Relocalization images are also saved to another text file that contains the orientations of each image.

The saving and loading function are mainly built for debugging purposes, since in

real scenarios the scene changes very often, so previously saved maps are inaccurate. This functionality could also be used for scenarios where the panorama image is created beforehand, for example, with an 360-degree camera. Using previously built panorama images allow creation of error-free maps.

6.4.1.6 Reinitialization

As with all visual trackers, the tracking is sometimes lost, if the camera moves too quickly or something obstructs the camera view. In these cases, it is important to have a system which can reinitialize the tracking back to a correct position as soon as possible.

The reinitialization system is closely based on the system described by Wagner et al. [17], and the relocalizer system is based on template matching. Every time the tracker has recognized more than 10 degrees of x-axis movement, the current frame is saved as a relocalization image. The selected image is first resized to 80x60 pixels in order to make the template matching in relocalization faster. The resized images are also blurred, to lower the amount of detail in the image. The selected images are then saved to a vector in the application with knowledge about the orientation of the selected images. Since the images are blurred, they are easier to match to whenever the relocalization is required. If something small has changed in the scene, template matching will not match the images correctly. Blurring the images makes the relocalizer functional even when small changes have occurred in the scene. If the memory usage of the system becomes too large, or the relocalization becomes too slow, the relocalization images can be made even smaller, and they can be taken at smaller intervals. The performance of the reinitialization system also drops with respect to the image count, since each frame of relocalization, all of the relocalization images have to be matched against the current frame.

When tracking is lost, the relocalizer begins running every frame. Each input frame is resized and blurred similarly to the relocalizer images, and then the downscaled current frame is matched against each saved relocalizer image. Whenever the template match

finds a match with good enough quality, it reads the current pose from the relocalizer images data. The viewpoint is then moved to that position, and normal tracking continues.

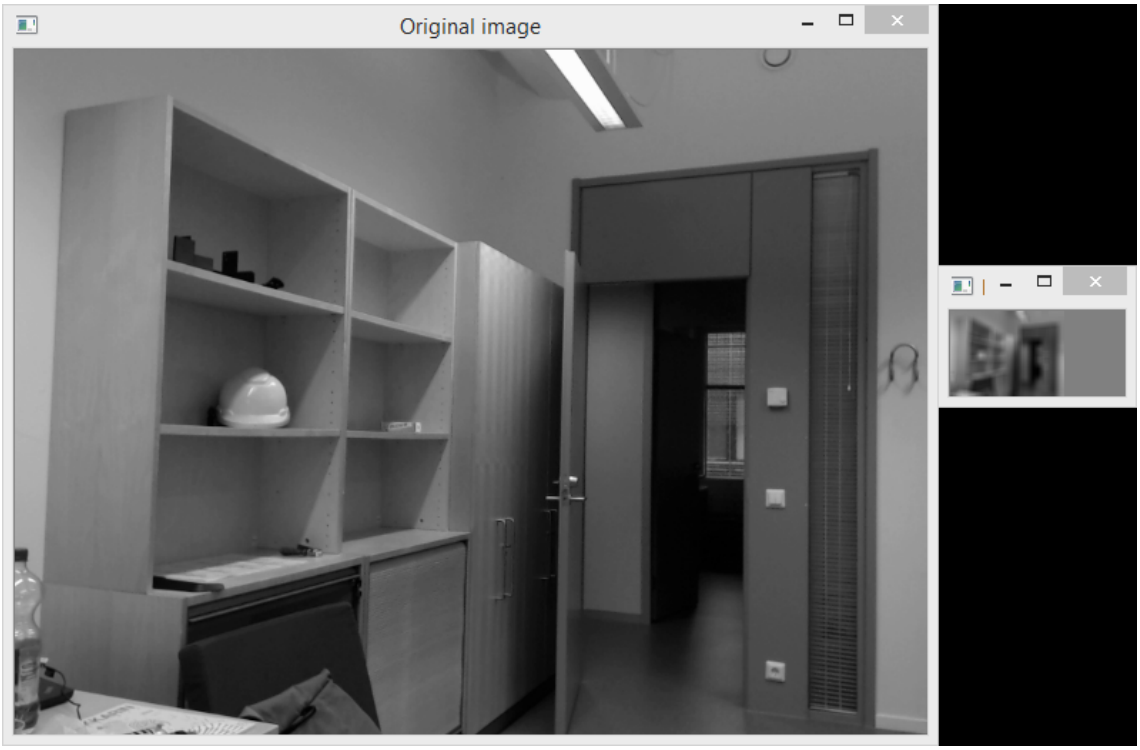

Figure 6.6: Sample image from the relocalizer. On the left the original image, and on the right the resized and blurred image that is used with reinitialization.

Chapter 7

Results

The results and findings of each developed application are presented in this chapter. The results are split into four different sections, one for each developed application.

7.1 Wikitude SDK

The first use mode (IMUs only with GPS on) was highly reliant on the accuracy of the GPS. Since the used devices had a basic GPS sensors, the GPS accuracy was in range of around 10 meters. In order to augment content accurately, this accuracy is not sufficient. When the GPS was pinned to a pre-defined location to determine the accuracy of the rest of the sensors, it quickly became obvious that the accuracy of these sensors is not good enough for these kinds of visualization tasks. The visualization did not look smooth, and due to the issues with magnetometers the position of the augmented content jumped around a lot. The Wikitude SDK also had its own downsides due to being a commercial system. This means that there was no possibility to affect the actual tracking of the system, or see how it was built.

The Wikitude demo application was a confirmation that pure IMU data is not sufficient for high quality tracking, and it was easy to see that Wikitude could not achieve the targets set in the project. However, the Wikitude SDK still could be used for a number of different

tasks, where the accuracy requirements are not so high. For example, it could be used for navigation systems, since the inaccuracies are not so obvious from larger distances. The demo application augmented models only when the user was closer than 100 meters to the target, but navigation applications often shows augmented content that is much further away, so small jittering is not noticeable and a 10 meter error in GPS makes barely any difference.

7.2 Sensor fusion

For sensor fusion calculations some statistics were measured in order to see the difference between the raw magnetometer data and the combined data. The graphs in Figure 7.1 clearly show that the gyroscope makes the movement a lot smoother when compared to the compass only. Augmenting content using Unity 3D was also tested, but as could be determined from results of the Wikitude application, the accuracy of IMUs is not sufficient in cases where high accuracy is required.

7.3 Natural feature tracker

The difference between the tracking by plain sensor data versus the combined solution was clearly visible during the testing. The augmentation looked more appealing to the eye. However, the drift in the system was very large if the device was rotated, for example, 90 degrees in horizontal direction. 90 degree movement was enough to move the augmented object too much from the original position of the object.

7.3.1 Issues in the system

A major problem in this type of tracking is the accumulation of drift. The features that go off the screen are lost permanently since there is no reinitialization. This causes a drift whenever the user rotates the device a lot and new features are constantly searched for.

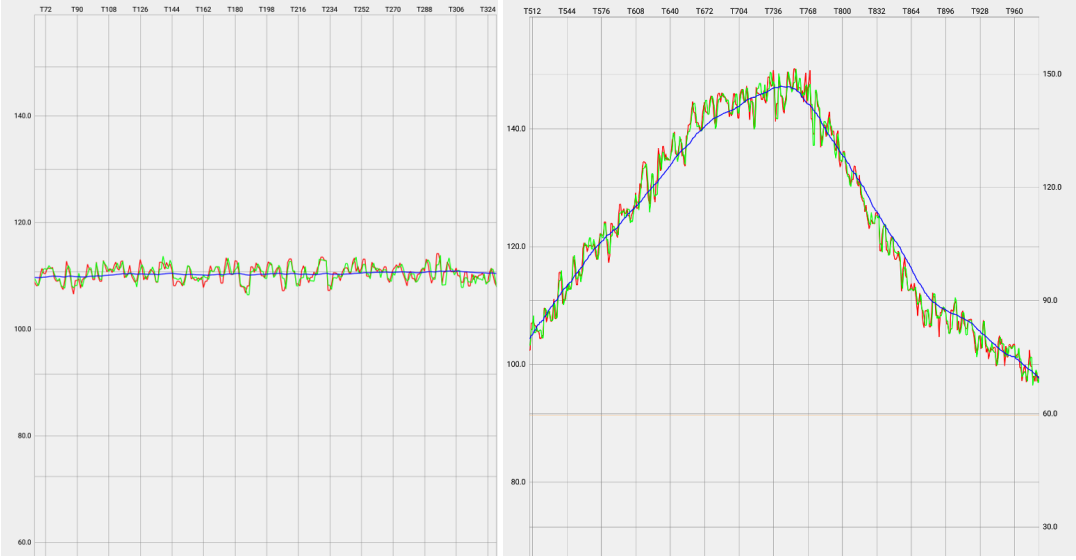

Figure 7.1: Compass orientation on y-axis, measurement times on x-axis. Blue represents the sensor fusion, red is the raw data and green is a sliding average of last measurements. The data in the left image was recorded when the device was steady, and the data on the right image was recorded when the device was rotated slowly.

If the device is not rotated very much and the original features remain visible, the drift accumulated is not that large.

Another issue with calculating the optical flow is the requirement for processing power. The processing power requirement becomes a problem especially since the program is supposed to be run on a mobile device such as a tablet. By lowering the amount of tracked features the performance problems can be compensated for to some degree, but using lower number of features also makes the tracking more inaccurate and causes more drift.

As it is with all visual trackers, it is also important to have a large enough number of easily distinguishable feature points (for example, some walls that have barely any texturing are impossible to track). Some terrains such as gravel can also cause some challenges, since there can be a large number of differences in gradient that make those spots good features, but since there are many similar spots, errors can happen in the

tracking of the features.

7.3.2 Possible improvements

The biggest future improvement for NFT would be to build some sort of reinitialization system, which would help to alleviate the accumulated drift in the system. For example, a reinitialization system that is used in the panorama tracker described in Section 6.4.1.6 could be used. The smoothness of the tracking could also be improved by some filtering. It could also be taken into account that a faster method for finding the features could be used such as FAST.

The drift compensation system could also be improved. In the current system the position is recalibrated with the magnetometer at static intervals, but the recalibration is not necessary if the device has not rotated by a large amount and no original feature points are lost. In these cases the recalibration interval could be longer.

7.4 Panorama tracker

After testing the panoramic tracker in multiple places both outdoors and indoors, it has become apparent that this type of tracking can potentially function in both scenarios. Originally, there was doubt about the quality of indoor tracking, since the distances between the camera and features are small, which cause issues since the translation is not taken into account. It was thought that involuntary translational movement when rotating the device would be problematic, but while holding the tablet steady, the translational movements were not that large. Indoor scenarios often include a lot clearer features for tracking, since the gradient differences are often much clearer in man-made environments than outdoors. However, it is worth noting that the usefulness of the system for indoor tracking is much lower, since in indoor scenarios the user often needs to move around much more,more frequently which requires restarting the tracking more often.

A set of 30 videos was recorded outside, from which 10 were 90-degree movement videos, 10 180-degree movement videos and 10 longer videos with random movement in all directions. The tracker was able to get to the end of each video without the tracking dropping in 28 of the videos, and there were not any large visual errors in any of the maps as long as the rotation estimation around z-axis was disabled. In two of the longer videos the tracking was lost, but the relocalizer restored the tracking shortly. Some smaller errors were still visible, where the maps accumulate ”waves” whenever the map is updated with a small error in the pose. These small errors however had no effect on the accuracy of the tracking. With the rotation estimation around z-axis enabled, the tracker was still able to get to the end of the videos but the quality of the maps was very bad and large visual errors were visible. Since the videos were recorded with a tablet, the automatic image adjustment options were at default, which made the map look a bit distorted in some parts of the image. The lightning errors are visible in Figure 7.3, where it is visible when the camera has adjusted the settings automatically. These errors however did not seem to have noticeable effect on the quality of the tracking.

A possibility for augmenting content was implemented for testing in the Unity 3D version of the application. The quality of the tracking looked very good when the device was on a tripod, and as expected the system did not accumulate any drift except for the errors that happened during the mapping process. The augmentation looked smooth and stayed in its initial position well. When the device was handheld and the rotation around z-axis was also estimated, the tracking quality was significantly lower, and the maps often became very distorted and untrackable. This is probably due to the movements being more shaky when the device was handheld, and the low quality of rotation estimation around z-axis. If the estimation of rotation around z-axis was turned off, handheld quality improved, and became very close to the quality while the device was set on a tripod. However, it can be difficult to keep the device from rotating even a little around z-axis when moving the device around.

The inaccuracy in estimating the rotation around z-axis is probably due to the estimation only calculating the difference in rotation between last two frames, instead of comparing it against the map. Since the rotation does not take the map into account, the rotation estimation can accumulate drift which will lower the quality significantly. The quality of tracking was much better, when the z-axis rotations were not taken into account. Using the device handheld was also possible without estimating the rotation around z-axis, if the user avoids large rotations around z-axis.

7.4.1 Performance

The largest problem in the system on mobile devices was the performance. On Unity 3D PC-version the fps during tracking was at 35-40, which is sufficient for high quality tracking. However on tablets the tracker was only capable of 5 fps. After integrating OpenCV for Tegra SDK [55] with the system, the performance was improved to the required 15fps. The issue with using OpenCV for Tegra is that the optimizations only work on mobile devices using the Tegra chipset.

Optimizing the system to improve the system further is quite difficult. Currently, a majority of the processing goes to the template matching of the small images. This can be affected by either lowering the number of tracked features, or by lowering the resolution of the camera images, but lowering the settings also lowers the quality of the tracking. Some processing power is also consumed when the images from the camera are converted to a format read by OpenCV. The tracking alone can be run on the Nvidia Shield tablet with 20 fps, but using Unity 3D causes some overhead which lowers the fps to 15.

7.4.2 Future improvements

There are multiple possible future improvements for the system. One of the major challenges is the calibration of the cameras. Especially on tablets, it is often very difficult to turn off features such as auto-focus, auto-exposure or auto-gain. All of the automatic

Figure 7.2: Sample panoramic image created by the tracker

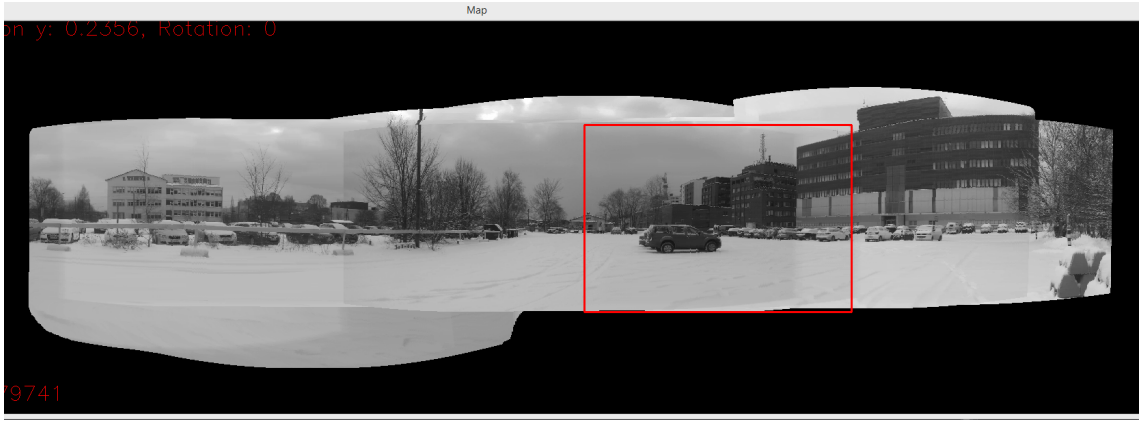

Figure 7.3: Sample panoramic image created by the tracker

image adjustments can cause major issues in tracking, since template matching might not find the keypoints correctly if the lightning or focus of the area is different. The problem with these automatic parameter adjustments is visible in Figure 7.3, where areas with different lightness can be seen very clearly. Vignetting, where the pixels in the edges of the camera are darker than in the middle, is also an issue which is unaccounted for. Another application is also required in order to accurately determine the actual field of view of the image.

The two biggest features for improving the quality of the maps is to take into account sub-pixel accuracies. Currently the result for tracking of a single frame might challenge that the change compared to previous frame is average of 2.5 pixels. This causes the issue of determining what to do when the result is not exact. Currently, the system works by simply rounding up the results for each frame, which cause visible distortions in some

parts of the map. The next improvement for the system is better handling of sub-pixel accuracies. The second large improvement is an improved method for pose estimation, which would eliminate any small jittering which can causes errors in the map creation. This type of filtering would also help a lot when the device is handheld instead of being steady on a tripod. These small errors can become very severe, since if a small error happens at some point of the map, that small error will disturb the tracking, which in turn will cause more errors.

The z-axis rotation estimation requires some improvements. Since the system currently compares two successive frames for estimating the rotation, drift accumulates. An improved estimation is required which estimates the rotation by comparing the current frame to the map, instead of the previous frame.

If the mapping process goes very wrong at some point, the map becomes useless for tracking purposes. In these cases, it would be important to have a feature in the application which could be used to update the maps after they are created, since now once a pixel is set, it will never be set again. This would be useful in order to account for small errors in the map, and possible changes in the scene or lightning.

Since the panoramic maps can be saved on PC, they should have an application with which one could edit and view the maps. Editing of the maps would be beneficial if, for example, some features on the map are located at a difficult-to-track positions. The saving and loading of the maps should also be extended to work on Unity 3D and Android tablets.

Chapter 8

Conclusions

This thesis presented a number of different methods that can be used for visual pose estimation outdoors. The idea for the thesis arose when it became obvious that there are currently no commercial solutions available, which could achieve the required accuracy. After the research of commercial solutions, a development for custom solution was begun. However, the initial custom solution had too large issues to be considered as a potential solution. After the initial solution, more research was done on potential outdoor tracking solutions. A paper presenting a panoramic tracker was selected for implementation as a potential solution for the presented issues.

The research into tracking technologies provided a large amount of insight on how the issues for outdoor AR could be overcome. It seems that the technology is already available, but no one has yet implemented these solutions for public use. Majority of problems have also been related to the hardware of the mobile devices, which have improved drastically in the last few years. If the hardware improvements continue, it is likely that the possibilities for outdoor pose estimation also improve.

Out of the developed solutions, the panoramic tracker seems like a potential solution for 3DoF tracking outdoors. Some issues still remain unsolved, but especially if the used device is held on top of a tripod the functionality can reach the required level in accuracy. The targets set for performance can also be reached, as long as the used device uses a

Tegra chipset so that the OpenCV for Tegra library can be used. Handheld use is also possible, if the rotation estimation around z-axis is turned off and the user is able to avoid large rotations around z-axis.

References

[1] Unity 3D. https://unity3d.com/.

[2] Microsoft. https://www.microsoft.com/microsoft-hololens/en-us.

[3] Robert Albrecht and Tapio Lokki. Auditory distance presentation in an urban augmented reality environment. volume 12, pages 5:1–5:19, March 2015.

[4] Katie Collins. Sensory hacking: perfume-infused dreams and virtual intimacy, 2014. http://www.wired.co.uk/news/archive/2014-03/31/touch-taste-and-smell-technology.

[5] Paul Milgram, Haruo Takemura, Akira Utsumi, and Fumio Kishino. Augmented reality: A class of displays on the reality-virtuality continuum. In *Proceedings of the SPIE Conference on Telemanipulator and Telepresence Technologies*, pages 282–292, October 1994.

[6] Siavash Zokai, Julien Esteve, Yakup Genc, and Nassir Navab. Multiview paraperspective projection model for diminished reality. In *Proceedings of the 2Nd IEEE/ACM International Symposium on Mixed and Augmented Reality*, ISMAR '03, pages 217–226, July 2003.

[7] Wikitude. https://www.wikitude.com/.

[8] TryLive. http://www.trylive.com/demos/trylive-eyewear/face-analysis.

[9] Rosemary Luckin and Danae Stanton Fraser. Limitless or pointless?; an evaluation of augmented reality technology in the school and home. *International Journal of Technology Enhanced Learning*, 3(5):510–524, August 2011.

[10] Wikitude. http://www.wikitude.com/showcase/wikitude-navigation/.

[11] Wikitude. https://www.youtube.com/watch?v=g-0cuqeUvCQ.

[12] Microsoft. https://www.youtube.com/watch?v=xgakdcEzVwg.

[13] Magic Leap. http://www.magicleap.com/.

[14] AR-media. http://www.armedia.it/i-mechanic.

[15] Ronald Azuma, Bruce Hoff, Howard Neely, III, Ronald Sarfaty, Michael Daily, Gary Bishop, Leandra Vicci, Greg Welch, Ulrich Neumann, Suya You, Rich Nichols, and Jim Cannon. Making augmented reality work outdoors requires hybrid tracking. In *Proceedings of the International Workshop on Augmented Reality : Placing Artificial Objects in Real Scenes*, IWAR '98, pages 219–224, 1999.

[16] Alberico Menozzi, Brian Clipp, Eric Wenger, Jared Heinly, Enrique Dunn, Herman Towles, Jan-Michael Frahm, and Gregory Welch. Development of vision-aided navigation for a wearable outdoor augmented reality system. In *Position, Location and Navigation Symposium - PLANS 2014, 2014 IEEE/ION*, pages 460–472, May 2014.

[17] Daniel Wagner, Alessandro Mulloni, Tobias Langlotz, and Dieter Schmalstieg. Real-time panoramic mapping and tracking on mobile phones. In *Virtual Reality Conference (VR), 2010 IEEE*, pages 211–218, March 2010.

[18] Gerhard Schall, Daniel Wagner, Gerhard Reitmayr, Elise Taichmann, Manfred Wieser, Dieter Schmalstieg, and Bernhard Hofmann-Wellenhof. Global pose estimation using multi-sensor fusion for outdoor augmented reality. In *Mixed and*

Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium, pages 153–162, October 2009.

[19] Jing Li and Xiangtao Fan. Outdoor augmented reality tracking using 3d city models and game engine. In *Image and Signal Processing (CISP), 2014 7th International Congress*, pages 104–108, October 2014.

[20] Steven Kerr, Mark Rice, Yinquan Teo, Marcus Wan, Yian Ling Cheong, Jamie Ng, Lillian Ng-Thamrin, Thant Thura-Myo, and Dominic Wren. Wearable mobile augmented reality: Evaluating outdoor user experience. In *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry*, VRCAI '11, pages 209–216, 2011.

[21] Søren Riisgaard and Morten Rufus Blas. Slam for dummies: A tutorial approach to simultaneous localization and mapping. Technical report, 2005.

[22] Chris Harris and Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

[23] Jianbo Shi and Carlo Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference*, pages 593–600, 1994.

[24] OpenCV. http://opencv.org/.

[25] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, volume 1, pages 430–443, May 2006.

[26] Edward Rosten and Tom Drummond. Fusing points and lines for high performance tracking. In *IEEE International Conference on Computer Vision*, volume 2, pages 1508–1511, October 2005.

[27] Edward Rosten. http://www.edwardrosten.com/work/fast.html.

[28] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *Computer Vision (ICCV), 2011 IEEE International Conference*, pages 2564–2571, November 2011.

[29] David Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference*, volume 2, pages 1150–1157 vol.2, 1999.

[30] Rafael Radkowski and James Oliver. *Virtual, Augmented and Mixed Reality. Systems and Applications*, chapter Natural Feature Tracking Augmented Reality for On-Site Assembly Assistance Systems, pages 281–290. 2013.

[31] Feng-Cheng Huang, Shi-Yu Huang, Ji-Wei Ker, and Yung-Chang Chen. High-performance sift hardware accelerator for real-time image feature extraction. *Circuits and Systems for Video Technology, IEEE Transactions on*, 22(3):340–351, March 2012.

[32] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *Proceedings of the 11th European Conference on Computer Vision: Part IV*, ECCV'10, pages 778–792, 2010.

[33] Bruce Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 674–679, 1981.

[34] Jean yves Bouguet. Pyramidal implementation of the Lucas Kanade feature tracker. *Intel Corporation, Microprocessor Research Labs*, 2000.

[35] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234, November 2007.

[36] Christopher Barthold, Kalyan Subbu, and Ram Dantu. Evaluation of gyroscope-embedded mobile phones. In *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference*, pages 1632–1638, October 2011.

[37] W.T. Fong, Son-Khim Ong, and Andrew Nee. A differential GPS carrier phase technique for precision outdoor AR tracking. In *Mixed and Augmented Reality, 2008. ISMAR 2008. 7th IEEE/ACM International Symposium*, pages 25–28, September 2008.

[38] Structure. http://structure.io/.

[39] Dell. http://www.dell.com/us/p/dell-venue-8-7840-tablet/pd.

[40] Texas Instruments. www.ti.com/sensortag.

[41] Variable, Inc. http://shop.variableinc.com/pages/frontpage.

[42] Nektarios Kostaras and Michalis Xenos. Assessing the usability of augmented reality systems. In *13th Panhellenic Conference on Informatics, PCI2009*, pages 197–201, October 2009.

[43] Epson. http://www.epson.com/cgi-bin/Store/jsp/Product.do?sku=V11H560020.

[44] Vuzix. https://www.vuzix.com/Products/LegacyProduct/5.

[45] Oculus Rift. https://www.oculus.com.

[46] Julie Carmigniani, Borko Furht, Marco Anisetti, Paolo Ceravolo, Ernesto Damiani, and Misa Ivkovic. Augmented reality technologies, systems and applications. *Multimedia Tools Appl.*, 51(1):341–377, January 2011.

[47] Bitstars. https://github.com/bitstars/droidar.

[48] doPanic. http://panicar.dopanic.com/.

[49] 3DAR. http://3dar.us/.

[50] BeyondAR. http://beyondar.com/.

[51] EPIC GAMES INC. https://www.unrealengine.com.

[52] Lenovo. http://shop.lenovo.com/us/en/laptops/thinkpad/t-series/t440p/.

[53] Nvidia. https://shield.nvidia.com/.

[54] Logitech. http://www.logitech.com/en-us/product/hd-pro-webcam-c920.

[55] Nvidia. http://docs.nvidia.com/gameworks/content/technologies/mobile/opencv_main.htm.