A Real-Time Mediated Reality Platform for Outdoor Navigation on Mobile Devices and Wearable Computers

by

Eric Tran

A thesis submitted in conformity with the requirements for the degree of Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

© Copyright by Eric Tran 2010
A Real-Time Mediated Reality Platform for Outdoor Navigation on Mobile Devices and Wearable Computers

Eric Tran
Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto
2010

Abstract

Wearable computing systems have been researched and developed for several decades. With the advent of the head-mounted display, augmented and mediated reality systems became an important example of wearable computing. However, due to certain factors such as computational constraints, cost, obtrusiveness, practicality, and social acceptance, mediated reality systems have been leveraged in only very specific application domains and have yet to see mainstream adoption.

This dissertation describes the research and development of a real-time mediated reality platform developed for modern mobile devices to provide a more reasonable transition in overcoming the mainstream adoption barrier of mediated reality systems. In particular, an outdoor navigational application that provides contextually-relevant information about a user’s surroundings is developed using the platform as a proof-of-concept for evaluation. In addition, the server infrastructure required to support the application is discussed, as well as the evaluation of a hybrid orientation tracking approach using sensors and computer vision.
Acknowledgments

My parents, for supporting me throughout my studies. I am where am I today because of them.

Professor Steve Mann, for his vision for the future and his drive to make the impossible, possible.

My best friend Ian Greig and his family, for being like a second family to me.

My close friends John Tzanakakis, Jason Weinstein, and Massimo Tarulli, for making Engineering memorable.

Raymond Lo, for all of the hacking sessions over the years. The stuff that we were able to accomplish still amazes me.

All of my students I’ve taught over the past couple of years, for giving me the opportunity to share my knowledge and sense of humour with them.
# Table of Contents

List of Tables ................................................................................................................................ vi

List of Figures ................................................................................................................................ vii

List of Appendices ........................................................................................................................... viii

1 Introduction .................................................................................................................................. 1
   1.1 Motivation ............................................................................................................................... 2
   1.2 Research Goals ....................................................................................................................... 5
   1.3 Thesis Structure ..................................................................................................................... 6

2 Background and Related Work ................................................................................................. 7
   2.1 Wearable Computing ............................................................................................................. 7
   2.2 Virtual, Augmented and Mediated Reality ......................................................................... 9
      2.2.1 Rendering ................................................................................................................... 10
      2.2.2 Tracking and Registration ......................................................................................... 11
   2.3 Chapter Summary ................................................................................................................. 16

3 Tracking and Registration Approach ...................................................................................... 17
   3.1 Overview ............................................................................................................................... 17
   3.2 Usage Scenarios and Assumptions .................................................................................... 17
   3.3 Approach ............................................................................................................................. 19
      3.3.1 Camera Position Tracking ....................................................................................... 21
      3.3.2 Camera Pitch Angle Tracking ............................................................................... 22
      3.3.3 Camera Yaw Angle Tracking ............................................................................... 23
   3.4 Performance Considerations .............................................................................................. 27
   3.5 Chapter Summary ................................................................................................................. 30

4 System Design and Implementation ...................................................................................... 32
   4.1 System Overview ................................................................................................................. 32
   4.2 Source Server ..................................................................................................................... 33
   4.3 Server ................................................................................................................................. 34
      4.3.1 Overview ................................................................................................................... 34
      4.3.2 Tiling and Layering of Geo-Data .............................................................................. 35
      4.3.3 Tile Database Format ............................................................................................. 37
List of Tables

Table 5-1: Test Data Descriptions ........................................................................................................ 50

Table 5-2: Average execution times with image down-sampling ..................................................... 56

Table 5-3: Average execution times with specifying a region of interest ......................................... 60
List of Figures

Figure 3-1: Overhead view of typical usage scenario ....................................................... 18
Figure 3-2: iPhone accelerometer axis alignment and tilt-angle determination. ............ 22
Figure 3-3: Overhead perspective of rendering a virtual scene during camera rotation. 25
Figure 3-4: Overhead perspective of rendering during camera rotation and translation. 26
Figure 4-1: General system overview diagram............................................................... 32
Figure 4-2: Client bounding box overlap issue............................................................... 35
Figure 4-3: Tile downloading and rendering................................................................. 40
Figure 4-4: Virtual environment rendering.................................................................. 43
Figure 4-5: Birds-eye view of virtual environment with compass rendering. ............ 45
Figure 4-6: Highlighting of building within user’s field-of-view. .................................. 46
Figure 4-7: Client running on an iPhone 3G. ............................................................... 46
Figure 5-1: Test setup used for data acquisition............................................................. 49
Figure 5-2: Yaw angle estimation with image down-sampling for Test 1A .................... 52
Figure 5-3: Yaw angle estimation with image down-sampling for Test 1B .................... 53
Figure 5-4: Yaw angle estimation with image down-sampling for Test 2A .................... 54
Figure 5-5: Yaw angle estimation with image down-sampling for Test 2B .................... 55
Figure 5-6: Yaw angle estimation with region of interest for Test 1A ......................... 57
Figure 5-7: Yaw angle estimation with region of interest for Test 1B ......................... 58
Figure 5-8: Yaw angle estimation with region of interest for Test 2A ......................... 59
Figure 5-9: Yaw angle estimation with region of interest for Test 2B ......................... 59
List of Appendices

Appendix A: Example OpenStreetMaps XML Data........................................71

Appendix B: SQLite Geo-Database Structure.............................................72

Appendix C: iPhone 3G Raw Camera Frame Acquisition.............................73
1 Introduction

A wearable computer can be described as a fully functional, self-powered, self-contained computer that is worn on the body [1], allows for general purpose processing, and has both operational and interactional constancy (i.e., it is always on and always ready/accessible) [2]. Wearable computing facilitates a new form of human-computer interaction where humans and computers are inextricably intertwined, operating synergistically in the same feedback loop [3]. This notion of synergistic operation is the key premise behind humanistic intelligence, where it is recognized that the human brain is perhaps the best neural network of its kind and instead of treating it as a separate entity and trying to emulate human intelligence in the computer (as is the goal with Artificial Intelligence), the computer should work in close synergy with the human [4]. This is in contrast to other devices such as personal digital assistants and smart-phones where they are wearable and operationally constant, but not interactionally constant: one still has to take the device out of their pocket to see the display or provide input to it.

An important example of wearable computing and synergistic operation is mediated reality [5], where information that humans would normally sense from the natural environment is modified to contain extra computer-generated information in real-time (i.e., the computer mediates what the human senses). For instance, using a transparent head-mounted display and auxiliary sensors, computer-generated graphics or text can be overlaid onto the environment that a human sees in order to provide additional contextual information or to enhance the visual aspects of what the human sees. Through long-term adaptation as a human wears such a system and uses it in their daily life, the system begins to function as an extension of their mind and body.
Wearable computing and mediated reality systems can generally be thought of as personal information devices [6]: they have been primarily designed to allow people to view/access, interact with, and manage information anywhere and at any time. This allows for many potential opportunities for improving the efficiency and quality of human labors, such as engineering, manufacturing, construction, diagnostic, maintenance, monitoring, and transactional activities [1]. Some wearable computing/mediated reality systems have been used to aid in medical visualizations [7], improve manual aircraft manufacturing processes [8], provide extra information for soldiers in military-related applications [9], and act as navigation systems for the visually impaired [10]. Other wearable computing systems and applications have been developed for enhancing vision as well as serving as a memory prosthetic for people who suffer from memory loss [11][12].

1.1 Motivation

Although there are many applications for wearable computing and mediated reality systems, the technologies have not yet seen mainstream adoption. There are many potential benefits that wearable computing and mediated reality systems could provide to the mainstream public such as being able to overlay building names/addresses on top of the actual buildings for navigational purposes or being able to display contextually-related information such as reviews and/or ratings for a restaurant a person is about to enter. However, due to certain factors such as computational constraints, cost, obtrusiveness, practicality, and social acceptance, wearable computing and mediated reality systems have been leveraged in only very specific or niche application domains and have yet to see mainstream adoption. For instance, some applications perform computationally-intensive operations such as image processing and transcoding which require computational resources and hardware not yet widely accessible in terms of cost and
availability, or suitable for mainstream use in terms of size or form factor (e.g., mobile workstations combined with other mobile equipment in the form of a backpack [13][14][15]). Other applications require a head-mounted display and/or external sensor hardware which are both typically obtrusive [16] and/or socially unacceptable for mainstream use.

However, due to recent advances in computing technologies, mobile device technologies, and the pervasiveness of mobile devices, wearable computing and mediated reality systems have become more feasible for mainstream applications. Processors have increased in performance while decreasing in size and power consumption, mobile devices have become more capable of running wearable computing applications with integrated sensors (e.g., accelerometers, video cameras) and wireless connectivity, and many people have adopted the use of mobile devices in their daily lives.

Head-mounted displays have also improved in terms of size and aesthetics but due to their cost as well as the need to wear them constantly, people may be less inclined to use them in their daily lives. Because of this, a compromise or trade-off between interactional constancy and obtrusiveness/social acceptance should be made in order for mediated reality applications to be widely adopted for mainstream use. This trade-off can be made with current mobile devices with integrated video cameras: instead of having a person wear a head-mounted display, they can simply hold the mobile device as if they were recording a video and view the computer-generated graphics or text overlaid onto the video signal rendered on the mobile device’s screen. This essentially provides the same basic functionality as a head-mounted display but allows for interactional constancy to occur only during the time when a person wishes to use the application. Using this approach could be the first steps towards mainstream wearable computing and mediated reality: it provides a more reasonable transition to overcoming the
social acceptance barrier by leveraging a platform that is accessible to the mainstream public and
does not force users to significantly change their day-to-day activities in order to obtain the
benefits of wearable computing and mediated reality technologies. This could allow for
increased adoption and awareness of the benefits of wearable computing and mediated reality
applications, which in turn may lead to future applications that might make people more inclined
to wear head-mounted displays.

A real-time platform or framework that leverages a mobile device to mimic the basic
functionality of a head-mounted display should be developed. The platform is not likely suitable
for every wearable computing or mediated reality application, but there are many applications
that could leverage it. For instance, navigation systems (e.g., the names of roads/streets within a
person’s field-of-view could be displayed so that users do not need to be at an intersection to
view the street signs), social networking applications (e.g., virtual messages could left on
buildings for acquaintances to find out where they must go), and personal safety applications
(e.g., significant events in a person’s life could be captured, monitored, and analyzed) could all
leverage the platform. In addition to helping break the social acceptance barrier (i.e., mainstream
adoption) for wearable computing and mediated reality applications, having such a platform
would allow for even greater personal empowerment than what current mobile devices provide:
the ability to bring contextually-relevant information to the user instead of the user searching for
contextually-relevant information. Furthermore, having such a platform would allow for the
possibility of rendering the contextual information in a more meaningful way (e.g., information
overlaid onto entities in the environment that the user actually sees as opposed to an abstract
representation, such as a map).
1.2 Research Goals

The overall goal is to research and develop a real-time platform or framework for wearable computing and mediated reality on a modern mobile device to mimic the basic functionality of a head-mounted display. The platform would provide the capability of running wearable computing and mediated reality applications without having users actually wear the device as they would a head-mounted display or other external sensors. The first objective is to determine whether this platform can be implemented in real-time on a modern mobile device (such as the Apple iPhone 3G). The second objective is to determine whether it is feasible for mainstream usage and adoption.

In order to demonstrate and evaluate the platform, a proof-of-concept outdoor navigational application will be developed that allows a user to view building information (e.g., names, addresses) as well as points of interest and streets overlaid on top of what they see using the camera of a mobile device. This would allow for users to simply aim the mobile device at a building, point of interest, or street to help them navigate to their destination. This is in contrast to searching for information about the user’s vicinity on a map and then orienting themselves to their surroundings to provide an internal mapping between the abstract representation of the world and the real world. This application might provide a more natural and intuitive method of navigating around a city. For instance, a user could see spatially-located text and images placed directly above points of interest or buildings as well as a path to travel to their destination without additional cognitive load. The implication of developing this application as well as the platform is that an appropriate orientation tracking method must be investigated to trade-off performance and accuracy.
Although advancements in mobile devices have progressed rapidly over the past several years, there are still device limitations and constraints that pose challenges in implementing mediated reality in real-time on a mobile device. For instance, the Apple iPhone 3G has computational limitations (e.g., processing speed and memory constraints), integrated sensor limitations (e.g., camera resolution and frame rate), and battery life limitations. The limitations and constraints and how they affect the real-time performance and feasibility for mainstream use will be investigated.

1.3 Thesis Structure

This dissertation consists of seven chapters: Chapter 2 presents the necessary background information, terminology and related work in the areas of wearable computing, mediated reality, and outdoor navigation applications. Chapter 3 discusses the approach used for tracking and registration and performance considerations that affect the accuracy and the computational run-time of the approach. Chapter 4 discusses the system design and implementation of the navigation application that uses the tracking and registration approach discussed in Chapter 3. Chapter 5 provides evaluation results for the tracking/registration accuracy, run-time performance optimizations, and usability of the system. Chapter 6 provides a high-level summary of the research contributions, results and findings. Chapter 7 discusses future work and applications.
2. BACKGROUND AND RELATED WORK

2.1 Wearable Computing

A lot of research and development has been previously done in the area of wearable computing. The first wearable computer was said to have been conceived in 1955 to predict the outcome of roulette [17]. With the advent of microprocessors and advances in computing technology starting in the 1970s, more general purpose wearable computers were developed which resulted in more applications for wearable computing. For example, a wristwatch was engineered by Mann to run GNU Linux and function as a videophone in 1998 [18]. It had the ability to transmit live video to the Internet at seven frames per second in full 24-bit colour with the aid of a separate device worn on the body.

An important application of wearable computing however, came with the advent of head-mounted display and head-up display technologies [19]. For example, with head-mounted displays, personal imaging became an application for wearable computing. With personal imaging, visual filters (e.g., freeze frame or real-time magnification) could be applied to assist the visually impaired. In addition, homographic modeling could be performed to enable the ability to add virtual text such as name tags in the user’s field of view with the correct perspective transformations [18][20].

The U.S. military also found uses for wearable computing and head-mounted display technologies with the research and development of its Land Warrior system [9]. The system consists of a helmet subsystem, control unit and communication subsystem, weapon subsystem, computer subsystem, and a navigation subsystem. The helmet subsystem houses hearing devices, microphone devices, and a head-mounted display which allows the soldier to interface
with Land Warrior features such as viewing his/her location, other friendly soldier locations, and his/her direction of travel superimposed on a map rendered in 800 x 600 resolution [21].

Head-mounted displays also find use in medical applications such as surgical procedures. Using a head-mounted display, a surgeon can essentially “see through” a patient and passively drive a sensor-equipped tool to deliver treatment to a targeted site in real-time [22].

Automotive applications have also found uses for head-mounted display technologies. For engine maintenance and repair tasks, diagnosis is usually performed by interfacing with the engine control unit and interpreting status/error codes displayed on a screen. However, this method of diagnosis causes the engine part to which the diagnosis is applied and the result data to be spatially separated. To address this, researchers have developed a prototype head-mounted display system to display diagnosis results in immediate proximity of the engine part using markers placed in specific locations on the engine [23].

Navigational applications have used head-mounted display technologies. For instance, the 3DVN system utilizes a head-mounted display with a 3-degree of freedom head-tracker, a wearable computer, and a data glove for gestural input to provide indoor wayfinding information to the user in the form of three-dimensional wireframe visualizations [24]. Outdoor navigational applications have also been developed using head-mounted displays such as the one developed by Azuma et al [25]. Using a GPS receiver, digital compass and tilt sensor, three digital gyroscopes, a laptop PC, a hybrid tracking approach was developed to be able to display virtual labels above actual landmarks viewed by the user.
Many of the applications that leverage head-mounted displays implement mediated reality to provide the user(s) with additional contextually-relevant information for what they are seeing. For this reason, mediated reality is discussed in further detail in the next section.

2.2 Virtual, Augmented and Mediated Reality

Augmented reality combines three-dimensional virtual objects into a real environment in real-time to provide the illusory experience that the two coexist. This allows users to see supplementary information (i.e., information that they cannot sense) to aid their perception of the real world and to interact with it. This is in contrast to virtual reality, where the virtual environment replaces what the user perceives as the real environment. It is also in contrast to mediated reality, where in addition to adding objects to reality, there also exists the option of removing objects (e.g., to filter out unwanted sensory information).

Mediated reality systems typically consist of two subsystems: a rendering system to display the combined result of virtual and real environments for the user to perceive with their senses and a tracking/registration system to allow for proper aligning of virtual and real environments. In most mediated reality systems, the biggest challenge is with the tracking/registration system due to the fact that improper alignment of the virtual and real world would cause the illusion of the two worlds coexisting to be compromised [26].

Mediated reality systems typically employ the use of head-mounted displays. There are two primary technologies for combining real and virtual elements using head-mounted displays: optical and video technologies. Optical head-mounted displays are typically see-through and employ the use of optical combiners that are partially transmissive and reflective so that the user can see the real world as well as observe virtual images reflected from combiners from head-mounted monitors (e.g., Holmgren describes a head-mounted display that transmits 30% light
Video-based head-mounted displays on the other hand, are not see-through and employ the use of one or two head-mounted video cameras to capture the real world and one or two monitors mounted in front of the user’s eyes to view the video signal. One of the advantages of video-based head-mounted displays is that there is flexibility in compositing the final image and presenting it to the user (i.e., it allows for mediated reality). Another advantage of the video-based approach is that it allows for the sequence of captured images to be used for applying different tracking/registration strategies. For example, the sequence of digitized images can be analyzed using computer vision to extract motion and/or camera parameters. The optical approach on the other hand must rely on only inertial sensors or other external head-tracking technologies.

Since this research focuses on bringing mediated reality to mobile devices for mainstream adoption, the optical approach is not feasible. Thus, the focus will be on video-based tracking and registration systems for determining camera parameters, in addition to the rendering system for compositing the virtual elements for presentation to the user. The background and related research for these two systems will be discussed in the following sections.

2.2.1 Rendering

In order to display three-dimensional virtual objects on a two-dimensional screen, the three-dimensional representation of the objects must be transformed and then rasterized into a frame buffer. This is done using a rendering pipeline such as the one used in the OpenGL API, which uses a series of matrix transformations that are applied to every vertex of each object [28]. These transformations are expressed as follows: \( v' = (M_{vp}M_{pd}M_{proj}M_{mv})v \), where \( M_{mv} \) is the model-view matrix used to transform the original vertex in world coordinates into eye coordinates with respect to the virtual camera, \( M_{proj} \) is the projection matrix used to apply
perspective transformations to transform the vertices into the viewing frustum defined by the virtual camera, $M_{pd}$ is the perspective division matrix used to normalize each transformed vertex into device coordinates, and $M_{vp}$ is the viewport matrix to transform the normalized vertices into window coordinates for rasterization and rendering into a frame buffer. To view the matrices, refer to Appendix F of [28].

The matrix transformations essentially provide the ability to render a three-dimensional scene from the viewpoint of a virtual camera. The virtual camera parameters such as field-of-view, position, and orientation can be adjusted by modifying the model-view and projection matrices to allow for aligning the virtual environment with the real environment (i.e., matching the virtual camera parameters with the real camera) when compositing the frame buffer. Several approaches for how these parameters are determined are discussed in the following section.

2.2.2 Tracking and Registration

In mediated reality applications, the most basic and difficult problem is with registration: the objects in the real and virtual worlds must be correctly aligned in order to provide the proper user experience. The reason it is difficult is because there are many errors associated with registration. These errors fall into two categories, static errors and dynamic errors. Static errors occur when the user and the objects in the environment are motionless and are the result of optical distortion in camera/lens systems, errors in the tracking system, mechanical misalignments, and incorrect viewing parameters (e.g., field-of-view). Dynamic errors occur as a result of system delays from the time the tracking system measures the position and orientation to the time the updated images are presented to the user [26].

The other reason that registration is such a difficult problem is that even small registration errors are perceptible to the user, such as incorrectly aligning a virtual picture with a
real picture frame. However, some outdoor mediated reality applications do not require pixel-accurate registration as some indoor mediated reality applications. For instance, identification of points of interest or landmarks in a user’s close vicinity does not demand as high registration accuracy as performing a surgical procedure where incisions require millimeter accuracy.

Furthermore, registration is more difficult in outdoor environments than in indoor environments because the option of applying additional markers or aids (e.g., coloured dots) or having prior knowledge about the environmental geometry and object positions beforehand is not possible.

There are two feasible approaches for obtaining the parameters of the camera that captures the real environment (i.e., position, orientation) for registration and alignment with the virtual camera, in the context of outdoor mediated reality: sensor tracking (e.g., GPS, accelerometers and gyroscopes) and computer vision techniques. These approaches as well as hybrid approaches will be discussed in the following three sections.

2.2.2.1 Sensor Tracking

There are several sensors available for outdoor mediated reality tracking: Global Positioning Systems (GPS), accelerometers/tilt sensors, gyroscopes, and digital compasses. The main disadvantage that prevents them from being ideal solutions for outdoor mediated reality is their inaccuracy due to noise, update frequency and other factors.

Typical GPS receivers provide a user’s coordinates within 30 meters at a rate of about 1 Hz [29] and allow for indexing into a spatial-database (i.e., geo-database) for localization. However, GPS requires direct line-of-sight with a sufficient number of satellites to provide good results and may not work well near urban areas or terrain with hills/canyons.
Inertial sensors like the accelerometers and gyroscopes allow for determination of linear and rotational accelerations respectively, but suffer from drift error. This is due to errors accumulating from the integration of samples from the accelerometers/gyroscopes to obtain the positional/rotational displacements. However, accelerometers can be used as tilt-sensors if the sensor is undergoing very little motion: the acceleration vector will point in the direction of gravity, allowing for the tilt-angle of the sensor to be recovered.

Digital compasses provide absolute heading values useful for lateral rotations. Unlike gyroscopes, the result is not relative and therefore does not suffer from drift error. However, digital compasses are still susceptible to noise, namely disturbances in the Earth’s magnetic field. In one study using the Precision Navigation TCM2 digital compass/tilt-sensor, it was found that magnetic distortion varied significantly with time and location and caused for errors in the range of 20-30 degrees [29].

Many outdoor mediated reality implementations have used combinations of these sensors to determine head orientation. For example, Azuma developed a system using a TCM2 compass/tilt-sensor and three orthogonally-mounted gyroscopes to estimate the angular position and rotational rate of a head-mounted display and overlay identification labels for nearby landmarks [25]. Using sensor fusion and Kalman filtering with constain gain, registration errors of only approximately 2 degrees were achieved. Another example would be the Touring Machine developed by Feiner [30]: using a commercial head-mounted display with built-in orientation tracker consisting of a compass and an accelerometer, a wearable computer and GPS receiver, the Touring Machine overlays labels of building name overlays as a user explores an environment. The Nokia MARA research project also leverages compass and accelerometer
hardware to annotate buildings or points of interest, but uses a mobile device with an integrated video camera instead of a head-mounted display [31].

2.2.2.2 Computer Vision Tracking

Computer vision can be performed on image sequences captured from a video camera to estimate motion and camera parameters. One approach that has been used for indoor tracking is to place fiducial markers of known geometry/dimensions in pre-determined locations in the environment and perform image analysis to identify the markers in the image and recover the transformation matrices relating marker coordinates to camera coordinates (e.g., [32][33]). These approaches typically have reasonable accuracy and are robust during partial occlusion of markers and accelerated camera motion. However, marker-based approaches are not appropriate solutions for outdoor environments due to the fact that outdoor environments are unprepared and one cannot expect to have the ability to control or change the environment as they would with an indoor environment.

Another approach that has been researched involves applying an edge detection filter (e.g., Sobel filter) on the images to extract terrain silhouettes to act as cues for determining camera orientation [34]. Although this approach is markerless, it requires the height-map (i.e., digital elevation map) of the terrain to be available to perform matching against the extracted silhouettes. Furthermore, the initial orientation of the camera needs to be provided as an initial guess to reduce the sample space of possible matches. Model-based recognition approaches have also been researched using a similar approach. For example, one approach utilizes a three-dimensional city model where visible edges are globally matched to edges extracted from video frames [35].
Another approach is to perform feature detection and matching on the images to track their displacement across consecutive images to estimate camera motion [36]. For instance, one approach extracts planar surface homographies from the image sequences by first extracting and matching points of interest and then computing the homographies [37]. This approach yields essentially zero jitter but suffers from slight drift error due to accumulation of errors from previously calculated homographies for the previous frames.

Featureless approaches that estimate changes in spatial coordinates arising from changes in orientation also exist. One particular example is the VideoOrbits algorithm which uses algebraic projective geometry to estimate the projective coordinate transformation between any pair of images for unprepared environments with high registration [18]. However, the algorithm has been found to run at only 11 frames per second on a high-end machine [38], which makes it unsuitable for real-time performance on mobile devices with less computational resources.

2.2.2.3 Hybrid Tracking

Previous studies have indicated that currently no single technology provides a complete solution for outdoor mediated reality and that hybrid tracking (i.e., combining several tracking technologies) is the only feasible approach [29]. Although hybrid tracking increases system complexity and cost, it provides more robust results because it allows for the opportunity for one technology to compensate for or address weaknesses in another tracking technology. This sometimes results in more accurate tracking than when using each technology individually. For example, previous work has shown that using inertial gyroscope sensors in combination with a vision system allows for significantly reduced registration errors [39]. This was because the gyroscopes were able to increase the robustness of the vision system by providing inter-frame
prediction of camera orientation and the vision system was able to correct for accumulated drift errors from the gyroscopes. Other examples of hybrid approaches are listed in Table 1 of [39].

2.3 Chapter Summary

This chapter introduced some relevant background information and related research for wearable computing and mediated reality applications. In particular, mediated reality systems, tracking and registration approaches, and existing outdoor navigation applications were discussed. Chapter 3 discusses the tracking and registration approach implemented for the prototype on the iPhone 3G. Chapter 4 discusses the overall system design and implementation of the outdoor navigation application.
3 Tracking and Registration Approach

3.1 Overview

The system is implemented on an iPhone 3G which possesses three sensors that are applicable for the purposes of tracking and registration: a GPS receiver, an integrated 3-axis accelerometer, and a camera. A hybrid tracking and registration approach was developed using the GPS receiver for positioning as well as orientation calibration, the accelerometer as a tilt-sensor, and the camera for orientation tracking. This chapter discusses the hybrid tracking and registration approach as well as the usage scenarios and assumptions that are made.

3.2 Usage Scenarios and Assumptions

In order to develop a tracking and registration solution that is both simple and effective, the common usage scenarios must be investigated to reveal possible simplifying assumptions.

The typical usage scenario for the outdoor navigation application involves the user proceeding to a particular location, stopping and then aiming the iPhone using their arm to view the surrounding area through the screen of the iPhone. When an object of interest (e.g., building) appears within the field-of-view, the user holds the iPhone in a relatively motionless state to view information about the object of interest. This process is repeated for when the user explores another location or views another object of interest.

The assumption that can be made is that the iPhone’s camera viewing direction and the normal of its screen are essentially collinear with the viewing direction of the user’s eyes during the motion. Therefore, the user’s arm and the iPhone temporarily function in a similar manner as a head-mounted display. This is illustrated in Figure 3-1 below:
3. TRACKING AND REGISTRATION APPROACH

![Figure 3-1: Overhead view of typical usage scenario](image)

The top diagram depicts the user proceeding from their original location at vertex \( O \) to a target location at vertex \( E \). The user holds the iPhone at vertex \( C \) to view the environment that contains object \( X \). The image captured and rendered on the user’s iPhone screen contains object \( X \)'s rasterized representation \( X' \) and is depicted by the red dot on the image plane. The bottom diagram depicts the user moving the iPhone from vertex \( C \) to vertex \( C' \) along the circle formed by radius \( EC \). Essentially, this depicts the user aiming the iPhone towards the left. Notice that the location of the rasterized representation of object \( X \) undergoes a horizontal displacement on the captured and rendered image.

Although Figure 3-1 illustrates the typical usage scenario when viewed from overhead (i.e., only lateral rotations are depicted), the same premise extends to when viewing from the sideways perspective (i.e., to illustrate vertical rotations): the user would aim the iPhone up or down and the location of the rasterized representation of object \( X \) would undergo a vertical displacement on the captured and rendered image.

During typical usage, another assumption is that the user will hold the iPhone a fixed distance away from their eyes as they are aiming it at different targets to be able to view and interpret the rendered images on the screen (i.e., the distance between the iPhone and the user’s eyes remains relatively constant when undergoing orientation changes). This is analogous to
having the eyepiece of a head-mounted display remain a constant distance in front of a user’s eye(s), except that the distance is now larger and is constrained to be at most the length of a user’s arm.

Another assumption is that the user will not rotate the iPhone in the plane of its screen significantly when aiming (i.e., around the axis collinear with the viewing direction of the camera or the user’s eyes). In other words, the iPhone will remain essentially axis-aligned with the environment during orientation changes (i.e., the bottom edge of the screen remains approximately parallel with the ground). This is a reasonable assumption based on the fact that the user needs to view and interpret the text as well as other graphical information that may be overlaid on the image rendered on the iPhone’s screen while aiming.

The other assumption is that the user runs the application outdoors during the day. This is so that the GPS signal is available and so that the environment is adequately illuminated such that the images captured by the camera have sufficient detail in order for feature detection to yield more results. The approach makes no assumption about the scene content as it may contain both static and dynamic elements (i.e., occluding/moving objects). This implies that when the view is completely occluded by a moving object, the approach will most likely yield undesirable/incorrect results as is the case with other purely computer vision-based orientation tracking approaches.

3.3 Approach

The general premise behind the approach is to model the virtual environment with the same scale as the real environment and render it on top of the video frame from the same viewpoint and perspective as the actual iPhone camera that the user is holding. This allows for the addition of virtual objects in their correct location in the real world. For instance, this allows
for the ability to overlay coloured lines overtop of the actual streets visible within the field-of-view. This would also provide the ability to place floating three-dimensional text in front of a building within the field-of-view.

The virtual environment can be modeled using a geo-database containing spatially-indexed geometry such as streets/roads, points of interest, and buildings plotted using GPS technologies. Furthermore, the geo-database can also contain relevant data pertaining to the geometry such as building/street names, addresses, or information to be able to perform a search query using a web service to retrieve other contextually-relevant information.

In order to estimate the position and orientation (i.e., yaw and pitch angle) of the camera, three of the iPhone’s sensors are used. Specifically, the GPS receiver is used to determine the camera’s position in real-world coordinates (i.e., latitude and longitude), the accelerometer is used to determine the camera’s pitch (i.e., tilt) angle, and computer vision is applied to the image sequence captured by the video camera to estimate the camera’s yaw angle.

To estimate the camera’s field-of-view, initial calibration methods can be used to determine the camera’s focal length (e.g., [40]), from which the field-of-view can be determined using: 
\[
\alpha = 2 \arctan \left( \frac{d}{2f} \right),
\]
where \(f\) is the focal length and \(d\) is the width of the image plane [41]. Alternatively, manual methods can be used to determine the camera’s field-of-view by using an object of known dimension \(d\) placed at a known distance \(f\) away from the camera such that the image acquired occupies the entire field-of-view. It is assumed that the field-of-view for the iPhone’s camera is known and provided as a constant.
The three camera parameters and their estimation approaches are discussed in further detail in their respective sections below.

3.3.1 Camera Position Tracking

The GPS coordinates obtained from the iPhone’s GPS receiver are used to index/query into the geo-database. The GPS coordinates and object coordinates in the geo-database are first transformed into Cartesian coordinates using a Mercator projection transformation given by:

\[
x = \theta - \theta_0, y = \frac{1}{2} \ln \left[ \frac{1 + \sin \phi}{1 - \sin \phi} \right],
\]

where \( \theta \) is the longitude, \( \theta_0 \) is the reference longitude, and \( \phi \) is the latitude. Both Cartesian coordinates are multiplied with the mean Earth radius (approximately 6,371,000 meters) to yield approximate real-world coordinates to allow for distance calculations (e.g., distance from one point of interest to another point of interest). This coordinate system is used for modeling the virtual environment and all of the objects within it.

The accuracy of the GPS coordinates obtained from the iPhone 3G fluctuates but is on average approximately within 30 ft (~10 m). Although the iPhone software development kit already implements accuracy improvement algorithms by leveraging positional data from multiple sources (e.g., Wi-Fi hotspots, cell towers), applying a Kalman filter to the GPS coordinates can possibly yield slightly more accurate results [42].

The vertical position of the camera is assumed to be constant and is a function of the user’s height. Automatic determination or calibration of this constant is out of the scope of this research and is left as future work.
3.3.2 Camera Pitch Angle Tracking

The accelerometer is used to determine the absolute pitch angle of the camera. This is because in the typical usage scenario, the assumption is made that the user will hold the iPhone steady once an object of interest appears within the field-of-view. During this stationary time, the accelerometer can be used to determine the direction of gravity. The tilt-angle can then be derived when comparing the direction of gravity with the mounting orientation of the accelerometer integrated within the iPhone. This is illustrated in below:

![Diagram showing iPhone accelerometer axis alignment and tilt-angle determination.]

Figure 3-2: iPhone accelerometer axis alignment and tilt-angle determination.
The illustration on the left indicates the axis-alignment of the accelerometer integrated into the iPhone. The illustration on the right shows the profile of an iPhone tilted forward with the acceleration vector of gravity plotted with respect to the y and z axes of the accelerometer.

Since it is assumed that the iPhone will not undergo changes in roll angle (i.e., rotations around the z-axis depicted in the left illustration in Figure 3-2), the tilt angle can be derived using only the y-axis and z-axis acceleration values: \( \phi = \pm \arccos \left( \frac{g \cdot (-\hat{u}_z)}{\|g\|} \right) \), where \( \phi \) is the
tilt-angle, $\mathbf{g} = (a_x, a_y)$, and $\mathbf{u}_y$ is the unit vector in the direction of the $y$-axis in the iPhone’s accelerometer coordinate system. Note that the tilt-angle is with respect to the horizon. As such, negative tilt-angles correspond to downward tilts and positive tilt-angles correspond to upward tilts. It can be noted that for downward tilts, the $z$-component of the gravity acceleration vector will project onto the $-z$ axis. Similarly, for upward tilts, the $z$-component of the gravity acceleration vector will project onto the $+z$ axis. Therefore, the sign of the tilt-angle can be determined by using the sign of the $z$-component of the acceleration vector.

The accelerometer data samples can be polled at rates of up to 100 Hz on the iPhone 3G [43]. The samples can fluctuate very suddenly and is accentuated with higher polling rates. The result of this is that the tilt-angles that are derived from the acceleration samples can undergo rapid changes, which is undesirable for tracking and registration as it would cause disorientation for the user. In order to avoid this, a simple low-pass filter can be applied to each axis of the accelerometer to respond slowly to sudden, short-lived changes in acceleration: $a = ka_i + (1-k)a_{i-1}$, where $k$ is the filtering factor, $a_i$ is the current acceleration value, and $a_{i-1}$ is the previously filtered acceleration value. This will allow the filtered acceleration value to smoothly stabilize to a value close to the actual acceleration value. Choosing a value for $k$ for the purposes of camera tracking is a trade-off between response time and the smoothness of the transition. In the context of the navigation application, a smoother experience is favoured over faster response time. Knowing this, a low value of $k$ is used (e.g., 0.1). (Note that $k$ is determined empirically and varies with the polling rate of the accelerometer).

### 3.3.3 Camera Yaw Angle Tracking

The previous two camera parameters are determined using absolute frames of reference. In the absence of a digital compass (as is the case with the iPhone 3G), there is no absolute frame
of reference to determine the camera yaw angle directly. However, the camera yaw angle (i.e., heading) can still be approximated by combining an initial heading value as an absolute frame of reference with relative offset changes in heading using relative frames of reference. This can be achieved by using the GPS heading to provide the initial heading and computer vision tracking to provide the relative rotation changes in camera yaw angle.

In order to determine the relative changes in yaw angle, feature detection and matching can be performed on consecutive video frame pairs to first determine the horizontal displacement of corresponding features in image space (e.g., using SURF [44]). Using the horizontal displacement, we can determine the rotation of the camera by treating the image acquired from the camera as if it was the output from rendering an image using a graphics pipeline such as OpenGL. For example, consider a virtual environment with right-handed coordinate system as the one used by OpenGL, in which the virtual camera is currently aiming at an arbitrary object of interest. Assume that the object of interest gets rasterized into pixel \((x_i, y_i)\) of the output image when rendered. Now assume that the virtual camera rotates by some arbitrary angle and the virtual environment is rendered again. The object of interest is now rendered at a new location \((x'_i, y'_i)\) in the output image. This is illustrated from an overhead perspective in Figure 3-3 below:
3. Tracking and Registration Approach

Figure 3-3: Overhead perspective of rendering a virtual scene during camera rotation. The illustration on the left shows a camera at vertex C rendering a scene consisting of object X onto the image plane. The rasterized location of object X on the image plane is indicated by vertex X’. The illustration on the right shows the camera rotating left by an arbitrary angle and rendering the same scene onto the image plane. The rasterized location of object X on the image plane is indicated by vertex X’’. The previous rasterized location of object X at vertex X’ is also shown. Notice the horizontal displacement of the rasterized location of object X after the camera rotation.

It can be seen that in terms of rendered output, this is analogous to having the camera remain fixed and instead, having the object of interest move from its original location \((x, y, z)\) to a new location \((x’, y’, z’)\) in the virtual environment such that when it gets rendered, it is rasterized into the same pixel location \((x_i’, y_i’)\) on the image plane.

Recall that every vertex gets transformed into image space coordinates by multiplying the original virtual world coordinates of a vertex by the compound matrix \(M = M_{vp} M_{pd} M_{proj} M_{mv}\). Knowing this, the camera’s yaw rotation can be calculated by first applying the inverse transformation \(M^{-1}\) to the rasterized locations of the object of interest before and after the camera rotation to yield their original world coordinates \(v\) and \(v’\) respectively:

\[
(x, y, z, w)^T = M^{-1}(x_i, y_i, z_i, 1)^T \Rightarrow v = (x/w, y/w, z/w)
\]

\[
(x’, y’, z’, w’)^T = M^{-1}(x_i’, y_i’, z_i’, 1)^T \Rightarrow v’ = (x’/w, y’/w, z’/w)
\]
3. TRACKING AND REGISTRATION APPROACH

Where \( y_i = z_i = y_i' = z_i' = 0 \) since only horizontal displacement in image space is considered.

Assuming the virtual camera is located at world coordinate \( \mathbf{c} = (x_0, y_0, z_0) \), the yaw rotation angle can be determined by projecting the two vectors \( \mathbf{v}_1 = (\mathbf{v} - \mathbf{c}) \) and \( \mathbf{v}_2 = (\mathbf{v}' - \mathbf{c}) \) onto the \( xz \)-plane and calculating the angle between the two vectors: \( \Delta \phi = \arccos \left( \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_1\|\|\mathbf{v}_2\|} \right) \).

However, since the user is holding the iPhone in their hand, the camera does not undergo pure rotation. The camera will undergo translation as well, since the user is in fact rotating about their torso with the iPhone at a fixed distance from the center of rotation. This is depicted in Figure 3-4 below:

![Figure 3-4: Overhead perspective of rendering during camera rotation and translation.](image)

The illustration on the left shows a user at vertex \( E \) holding the iPhone camera at vertex \( C \) a fixed distance \( r \) away from their center of rotation. The camera is rendering a scene consisting of object \( X \) onto the image plane. The rasterized location of object \( X \) on the image plane is indicated by vertex \( X' \). The illustration on the right shows the user rotating left by an arbitrary angle formed by vertices \( CEC' \) and rendering the same scene onto the image plane. The rasterized location of object \( X \) on the image plane is indicated by vertex \( X'' \). The previous rasterized location of object \( X \) at vertex \( X' \) is also shown. Notice the horizontal displacement of the rasterized location of object \( X \) after the camera rotation.

Therefore, the actual change in yaw angle that the virtual camera must be rotated by is the angle formed by vertices \( CEC' \) in Figure 3-4 above, not the angle formed by vertices...
X'C'X''. Since the world coordinates of the object of interest before and after the rotation (v and v') were calculated previously, the yaw angle can be calculated in a similar fashion to $\Delta \phi$, but using a position $c' = c - r\hat{v}_g$ instead of $c$ (where $r$ is the distance between the iPhone and the center of rotation and $\hat{v}_g$ is a unit vector in the gaze direction of the virtual camera) to form vectors $v_1$ and $v_2$. The implication of this is that $r$ must be estimated and as such, it is a potential source of error that can accumulate and lead to drift error.

The advantage of using this approach is that the perceived registration has the potential of being pixel-accurate since operations are performed in image space (similar to ray-tracing). This is provided that corresponding features can be matched and tracked from frame to frame. However, a possible source of error would be in determining the horizontal displacement of corresponding features matched in consecutive image pairs. Under perfect horizontal translation, the corresponding features would experience identical horizontal displacement. However under rotation, corresponding features will experience differing displacements. Furthermore, when occluding/moving objects are considered, not only will corresponding features experience different displacements, but they will possibly experience different displacement directions as well. Since this orientation tracking approach relies on relative rotational changes, choosing inaccurate horizontal displacements will cause drift error to occur.

The sources of error mentioned as well as other considerations that affect the performance of this approach are discussed in the following section.

3.4 Performance Considerations

The accuracy of the tracking approach described in the previous section is largely dictated by the performance of the camera yaw estimation. This is because under typical usage,
it is assumed that users will most often be performing lateral rotations to view the environment surrounding them. Furthermore, because the camera yaw estimation uses computer vision algorithms to perform feature extraction, it largely dictates the computational run-time of the overall tracking approach.

The total error in yaw angle accuracy that can be observed when a user is stationary and viewing the environment can be partitioned into two components: the static error and the dynamic error (i.e., $\varepsilon = \varepsilon_s + \varepsilon_d$). The static error in this context is the initial heading provided by the GPS receiver. In other words, no matter how accurate the relative yaw angle tracking approach is, there is at least a constant error offset $\varepsilon_s$ in the calculated heading due to the error in the initial heading. The dynamic error in this context is the accumulated drift error due to the relative yaw angle estimation. The total value of $\varepsilon_d$ is a function of three main factors in addition to the method used to determine the horizontal displacement from frame to frame: the number of features extracted, the spatial distribution of features chosen to determine the horizontal displacement, and the estimation of the distance between the camera and the user’s center of rotation.

The method used to determine the inter-frame horizontal displacement is significant in reducing the total dynamic error $\varepsilon_d$ observed. The ideal case that would yield the lowest error would be to track only one corresponding feature that is present for a significant portion of the rotation (i.e., the feature is present across many frames). Once that feature is no longer within the field-of-view, another corresponding feature is chosen where it would be tracked and so on. However, relying on only one feature decreases the robustness of the tracking approach in cases when that feature is no longer found (e.g., the feature was occluded by a moving object in the
scene). Therefore, multiple features must be considered to provide for robustness in outdoor, unprepared environments with dynamic scene content. The approach that is used to determine the global horizontal displacement from multiple features is to use the median horizontal displacement. An extension to this approach is to perform a weighted average using the median displacement and its nearest neighbours within a fixed size sampling window. Assuming the median is found at displacement $d_i$ in the sorted array of displacements $d_0$ to $d_N$, the global displacement can be calculated as:

$$d = \frac{k_{W/2}d_{i-W} + \cdots + k_id_i + \cdots + k_{W/2}d_{i+W}}{\sum_{i=0}^{W/2} k_i}$$

Where $k_0$ to $k_{w/2}$ are the weighting factors and $d_{i-w}$ to $d_{i+w}$ are the displacements within a sampling window of size $w$. The weight for $k_0$ should be 1.0 to favour the actual median.

It is generally desired to have a high number of corresponding features to increase robustness under different environment conditions, however extracting features comes at the cost of computation time as the entire image must be scanned. In addition to extracting features, matching features to identify feature correspondences across image pairs is also computationally expensive depending on the matching algorithm, the number of features extracted, and the size of the feature signatures generated by the feature extraction algorithm. Therefore, there is a trade-off between the number of features extracted and matched (i.e., computation time) and the accuracy of the tracking approach.

There are two general ways to reduce the computation time of feature extraction and matching that both involve reducing the number of features extracted by reducing the number of pixels considered: down-sampling the reference image and reducing the dimensions of the region.
of interest to perform feature extraction on. Down-sampling however affects the fidelity of the reference image, which in turn reduces the number of features extracted and the ability to generate unique signatures for features. Therefore the number of corresponding features is reduced.

Furthermore, reducing the dimensions of the region of interest to perform feature extraction on reduces the number of features as well as the number of feature correspondences. This is not only because of the reduced number of pixels scanned, but also because the probability of finding feature correspondences between image pairs within region of interest decreases as the size of the region of interest decreases. In other words, the likelihood that a feature detected in a region in an image $i$ will be detected in the same region in image $i+1$ is less if the region is small. This is especially the case if the images exhibit scene displacement.

The estimation of the distance between the camera and the user’s center of rotation also affects the calculation of the yaw rotation angle. This value is to be determined empirically as a distance at which users can comfortably view and interpret the images rendered on the iPhone’s screen. It is assumed that this focal distance does not vary significantly from person to person and is constrained by the average length of a human arm.

### 3.5 Chapter Summary

This chapter provided the background behind the approach used to determine the camera parameters as well as an overview of the factors affecting the approach’s performance in terms of accuracy and computational run-time that will be evaluated. Chapter 4 will discuss the system design and implementation of the navigation application that uses this tracking and registration method as well as the server infrastructure to support the application. Chapter 5 will provide
evaluation results pertaining to the accuracy and run-time performance of this orientation tracking method.
4 System Design and Implementation

4.1 System Overview

The system that was implemented consists of three main components: the client, the main server, and the source server. The client is the mobile device that the end-user interacts with to view and navigate their surroundings. The client dynamically requests geo-data for the area surrounding the user’s proximity from the main server to construct a virtual environment to render and overlay on top of the video frames captured from the mobile device’s camera. The client also performs orientation tracking to determine where the user is aiming the device. The main server stores binary-formatted geo-data and provides access to it via a web-service. The source server is an external third-party server that hosts the actual geo-data for the entire world and exposes an API to allow access to the data. The interactions between all three components are depicted in Figure 4-1 below:

![System Overview Diagram]

**Figure 4-1: General system overview diagram.**
The client uses its GPS coordinates to determine which area to request from the main server. The client makes the request to the main server via the internet, where the main server performs a check to determine whether the data for the request exists or not. If it exists, the main server sends the data to the client. Otherwise, it requests the data from the source server.
and reformats the data in a different format before storing it and sending it to the client. The client, upon receiving the data builds the three-dimensional representation of the environment and renders it along with the video frames obtained from its camera.

The three components are discussed in further detail in their respective sections below.

4.2 Source Server

The geo-database that is used is OpenStreetMaps, which is a spatially-indexed database containing user-generated street maps for the entire world [45]. Similar to Wikipedia, OpenStreetMaps allows users to create a set of map data that is free to use and edit. Research has shown that for heavily-populated areas such as London, England (i.e., areas with more OpenStreetMaps users), the coverage and accuracy of the OpenStreetMaps data is comparable to professional grade geo-databases (approximately 90% coverage) [46][47][48].

OpenStreetMaps data consists of only three simple primitives: nodes, ways, and relations. Nodes are essentially points consisting of a latitude and longitude coordinate. Nodes can be used individually to define points of interest. Groups of nodes can also be used to define other primitives such as ways. Ways are sequentially ordered interconnections of two or more nodes that essentially define linear elements such as streets/roads, or even building outlines or areas when the ways are closed loops. Relations are a higher-level grouping of nodes, ways or even other relations. For instance, a group of points of interest and areas (i.e., ways) may be described as one particular relation such as a university campus.

Each primitive has an arbitrary number of tags that contain extra data related to the primitive. The tags are essentially key-value pairs that can contain arbitrary Unicode strings. Certain key values are standardized for the format to indicate the primitive type (e.g., “highway” indicates a street or road). An example use for tags is to indicate a primitive’s name and address.
In general however, tags can be used for much more such as providing other contextual information such as customer reviews for a point of interest (e.g., a car dealership).

The source server that is used for the prototype implementation is hosted by OpenStreetMaps and exposes an API that allows for retrieving the geo-data in an XML format [49]. An example of the format can be found in Appendix A. The source server stores the geo-data in a spatially-indexed relational database that allows for faster retrieval of queries involving geometric data. For instance, the spatially-index database allows for queries to obtain all entities within a bounding box defined by Cartesian coordinates (e.g., GPS coordinates) in much faster time than with traditional indexing used by typical relational databases.

4.3 Server

4.3.1 Overview

The main server acts as an intermediate proxy and cache for OpenStreetMap data that has been converted into a format optimized for downloading and rendering on the client. This approach was chosen to prevent the need for querying the source server directly from the client for every single request made. It was also chosen to prevent the conversion of the data from being executed on the client for every single request made. This approach reduces overall amortized execution time and transmission latency by querying the source server and performing the conversion process only once; every subsequent request by the client for the same data from the server will simply be a file transfer of static data.

However, to allow for this caching approach to work for multiple clients, the requests must be normalized such that previously converted data can be reused/shared. For example, if one client (Client A) requests data for the area surrounding them and another client (Client B) is
located near that same area and makes a request, there will be some overlap in the data that is sent to them. This is depicted in Figure 4-2 below:

![Figure 4-2: Client bounding box overlap issue.](image)

Client A makes a request to the server for data for the area surrounding it, defined by Bounding Box A. Client B, located geographically nearby Client A, makes a similar request for data for the area surround it, defined by Bounding Box B. The data returned to Client A and B would contain duplicate data for the overlapping region.

This would result in uniquely converted data files for each request for the different clients, which would prevent reusing of previously converted data amongst multiple clients. To address this, the entire geo-database is partitioned into tiles and layers. The tiling and layering approach is discussed in further detail in the following section.

### 4.3.2 Tiling and Layering of Geo-Data

The geo-database contains spatial data for the entire world, which in terms of latitude and longitude, consists of GPS coordinates within the range of -90 degrees to 90 degrees and -180 degrees to 180 degrees respectively. The general premise behind the tiling approach is to allow clients to index their current GPS location into the correct tile and download that particular tile only. When the users change their position, the client will download neighbouring tiles dynamically in a seamless manner. This allows multiple clients to reference the same tile data,
even though their GPS locations may differ slightly, allowing for reusing of previously converted (i.e., cached) data. This also has the added benefit of reducing the amount of data that needs to be downloaded to the client before it can begin rendering the virtual environment.

The tile dimensions were chosen to be 0.01 degrees (width) by 0.01 degrees (height). This represents an area of approximately 1.1 by 1.1 kilometers after applying the Mercator projection transformation outlined in Section 3.3.1, which results in the entire world being partitioned into a grid of tiles with 18,000 rows and 36,000 columns for a total of 648 million tiles. Since oceans cover approximately 71% of the entire surface area of the Earth [50], only approximately 188 million tiles would contain data in the worst-case.

The tile dimensions were chosen to roughly trade-off file size (i.e., working data set size) and frequency of downloading new tiles (i.e., when a user approaches the boundary of a tile and neighbouring tiles need to be downloaded). However, the actual amount of data contained within a particular tile will vary depending on the geographical region. Optimal tile dimensions under different usage scenarios and geographical regions are not investigated and are out of the scope of this research.

Layers on the other hand, partition the different categories of data. Every layer contains a grid of tiles spanning the entire Earth, but the data contained within the tiles pertain to a particular category such as streets, buildings, or points of interest. This layering approach allows for clients to request and render only the data they wish to view information about. For instance, a client may only be interested in building names and addresses and therefore should only download tiles from the layer that contains only building data. This also has the added benefit of allowing clients to render multiple layers simultaneously for added flexibility.
The current implementation only uses one layer to store static geographical data – namely, streets, buildings and points of interest. However, the data can be partitioned into appropriate layers at a later time if desired. Furthermore, other layers can be populated and added without much implementation effort. Physically, the layers on the server are simply directories containing the individual tiles. So in order to add another layer, a new directory just has to be created and populated.

4.3.3 Tile Database Format

OpenStreetMaps provides the geo-data in XML format which is not the most efficient format for querying on the client. For this reason, the server converts the XML data provided by the source server into an SQL database using the SQLite3 database engine. Each individual database represents a tile that is sent to the clients. The database structure used for the tile is described in Appendix B.

SQLite3 was chosen due to it being self-contained, serverless, performant, and the fact that its cross-platform code and endian-agnostic file format allows it to be executed on many processor architectures such as the ARMv6 processor used in the iPhone 3G. The binary format of the SQLite3 database also reduces the size of data being transmitted to the client in comparison to XML.

4.3.4 Tile Web-Service

The server is simply a web-server running Linux and Apache using the ReiserFS filesystem to support the large number of tiles. The server hosts a web-service that provides access to the tiles. The web-service is implemented as a Perl CGI script that requires a parameter \( t \) specifying the desired tile index. The tile index \( t \) is specified by the client during the request and is calculated as follows:
4. SYSTEM DESIGN AND IMPLEMENTATION

\[ t = \left[ \left( \phi + 90^\circ \right) / H \right] \times C + \left[ \left( \theta + 180^\circ \right) / W \right], \text{ where } H = W = 0.01 \text{ and } C = \left\lfloor 360^\circ / W \right\rfloor \]

The offsets of 90 and 180 degrees are to map GPS coordinates, which range from -90 degrees to 90 degrees and -180 degrees to 180 degrees for latitude and longitude respectively, to be zero-based when determining the tile index. The inverse operation to yield the bounding box coordinates of the tile is simply given by:

\[ \text{bottom} = i \times H - 90^\circ, \text{ top} = \text{bottom} + H, \text{ left} = j \times W - 180^\circ, \text{ right} = \text{left} + W \]

Where \( i = t / C, j = t \% C \)

The web-service simply checks if the tile with the specified index \( t \) exists. If the tile exists, it is served to the client. Otherwise, a dynamic request is made to the source server for the geo-data within the bounding box defined by \((\text{bottom, left})-(\text{top, right})\) and the returned XML data is converted into SQLite3 format.

4.4 Client

4.4.1 General Overview

The client is implemented as an iPhone 3G (OS Version 2.2.1) application utilizing OpenGL ES 1.1 to render the video frames from the iPhone’s camera and the virtual environment. The Oolong Engine [51] is used to provide the general application framework. The main application code is implemented in C++ and is invoked from shell code implemented in Objective-C. The video frames are acquired using an unofficially supported camera callback that is described in Appendix C.

The overall flow of execution of the client is partitioned into four stages, similar to that of a typical real-time game engine: initialization, sensor data acquisition, world update, and
rendering. During the initialization stage, the client initializes and configures the accelerometer, GPS and camera sensors using the iPhone SDK APIs. OpenGL is also initialized as well as the GUI elements to display the results on the iPhone’s screen. The OpenGL camera was set to use a field-of-view of 37 degrees and a viewing radius of 400 meters. The field-of-view of the iPhone’s camera was determined manually to be approximately 37 degrees and the viewing radius was chosen to be less than half of the minimum tile dimension (400m) to reduce the number of possible tiles within the camera’s field-of-view.

For the sensor configuration, the accelerometer polling frequency was set to 15 Hz. The GPS sensor was set to use the best location accuracy (kCLLocationAccuracyBest) and a distance filter of 1.0. The camera controller was set to enable auto focus without displaying the focus reticule and was binded to a UIView control. A scheduled timer is also configured to fire at a rate of 15 Hz, which executes the world update and rendering stages continuously until the user quits the application. This implies that the maximum frames per second that can be observed is 15 FPS. This was chosen to reasonably balance animation smoothness and processing power, so as to not occupy 100% of the CPU cycles available to allow other iPhone applications to execute.

Sensor data acquisition occurs asynchronously for the GPS and camera sensors via callback functions. The accelerometer samples on the other hand, are polled and filtered using the low-pass filter described in Section 3.3.2 with \( k = 0.1 \) before execution of the world update stage. The accelerometer samples are used to update the virtual camera pitch angle.

For GPS updates, the virtual camera position is updated by applying a Mercator projection transformation. For camera callbacks, the image buffer is updated and feature extraction and matching is performed to determine horizontal displacement. The horizontal
displacement is then used to determine the estimated yaw rotation angle as described in Section 3.3.3. Feature extraction and matching is discussed in further detail in Section 4.4.2.

In the world update stage after the sensor data has been polled and the results applied to update the virtual camera position and orientation, the virtual environment is then updated to reflect the changes. First, the new virtual camera position is used to determine the tile index for the geo-data in the client’s proximity. The new virtual camera heading in conjunction with the camera’s viewing distance and field-of-view is then used to determine which neighbouring tiles are within the client’s field-of-view and should be downloaded and rendered. This is illustrated in Figure 4-3 below:

![Figure 4-3: Tile downloading and rendering.](image)

The figure above illustrates the client, located at vertex E and the camera, located at vertex C in the virtual environment. The client’s current GPS coordinates are used to determine the tile that contains the client’s current location: Tile \((i, j)\). The camera’s heading, field-of-view, and viewing distance are then used to determine which other tile(s) are visible from the client’s location: Tiles \((i-1, j+1)\) and \((i, j+1)\). The tile that is grey is not rendered. Any tiles that are to be rendered but do not currently exist on the client are scheduled for download using a background thread.
4. SYSTEM DESIGN AND IMPLEMENTATION

The tiles are dynamically downloaded, loaded to memory, and unloaded from memory using background threads (pthreads) as the user moves throughout the environment to provide a seamless experience.

Rendering is the final step which simply involves compositing the rendering of the virtual environment on top of the current video frame. This is achieved by having the current video frame rendered using the UIView control it was binded to and having the virtual environment rendered to a separate GLView that is layered on top of the UIView. The virtual environment is rendered by rendering only the tile that the user currently belongs to as well as the tile(s) that are within the camera’s field-of-view.

4.4.2 Feature Extraction and Matching

To perform feature extraction, the SURF algorithm is used [44] because it is currently one of the most robust feature extractors available and is scale and rotation invariant. This was a requirement because outdoor tracking and registration requires robust feature extraction and signature (i.e., descriptor) generation for matching. The actual implementation of SURF is provided by OpenCV 1.1.0, which was ported to the iPhone by specifying `-marm -arch armv6` when building using Apple’s port of GCC: `arm-apple-darwin9-gcc-4.0.1`.

A Hessian threshold of 500 is used, with 64 element descriptors, 2 octaves and 2 layers as parameters to the SURF algorithm. The choice for 64 element descriptors as opposed to 128 element descriptors was made to reduce feature matching execution time, which is a function of both the number of features and the number of descriptors per feature. The number of octaves and layers was chosen to be 2 to provide reasonable scale invariance, given that the majority of the features would not exhibit scale invariance. This is due to the fact that it is assumed that the
user is stationary when aiming the camera around (i.e., the distance from the camera to the static objects within the scene remain essentially constant).

The Hessian detector used by SURF in the OpenCV implementation (cvsurf.cpp) is computationally expensive as the nested loop to calculate the Hessian values has to be performed $O*L$ times where $O$ is the number of octaves and $L$ is the number of octave layers specified. As such, the nested loop was unrolled twice to provide better cache locality for the ARMv6 and improved execution time. This was empirically determined and optimization analysis for the Hessian detector is left as future research.

To perform feature matching, a linear search to find the corresponding feature that has the most similar descriptors (i.e., nearest neighbour) is used. The similarity between two features $f_1$ and $f_2$ is determined by calculating the Euclidean distance between the vector formed by the features’ descriptors:

$$d = (f_{1,1} - f_{2,1})^2 + (f_{1,2} - f_{2,2})^2 + (f_{1,3} - f_{2,3})^2 + ... + (f_{1,N} - f_{2,N})^2$$

Where $f_{i,j}$ is the $j^{th}$ feature descriptor for feature $i$, and $N = 64$ (i.e., the number of descriptors). A linear search was chosen over a kd-tree as kd-trees are not suitable for efficiently finding the nearest neighbours in high-dimensional spaces [52].

To perform horizontal displacement determination, the difference between each of the corresponding features’ x-components in image space is calculated and sorted into an array. The median element of the array is the horizontal displacement used for yaw rotation estimation.
4.4.3 Virtual Environment Rendering

The virtual environment is rendered using OpenGL line and triangle strip primitives. To annotate the buildings and points of interest in the virtual environment, bitmapped fonts are rendered to transparent quads and positioned 4 virtual meters above the ground at their correct location. In the case of points of interest, the bitmapped fonts are positioned at the actual coordinates of the points of interest. In the case of buildings however, the center of the building footprint is approximated to be the average coordinate of all nodes defining the building’s footprint (i.e., way). To increase the visibility and provide depth cues for the annotations, arrows are rendered as markers below the quad. This is illustrated in Figure 4-4 below:

![Figure 4-4: Virtual environment rendering.](image)

The above figure is a rendering of the virtual environment at the intersection of King’s College Road and College Street in Toronto, Canada. The video frame that would normally be rendered in the background is removed for clarity. The building footprints are rendered as green line strips. The streets and roads are rendered as thick yellow triangle strips. The building names are rendered above the approximated center of their respective building footprints, oriented so
that the text plane is orthogonal to the camera’s viewing direction. The street names are rendered above the streets, aligned with the street directions.

The orientation of the quads is made such that the plane of the text is orthogonal to the camera’s viewing direction so that the text is always legible. The annotations for streets and roads on the other hand, are aligned with the actual streets themselves and are positioned to be above the roads at points within the camera’s field-of-view. This is also illustrated in Figure 4-4 above, where College Street and King’s College Road intersect.

Because there may be many entities with annotations within the client’s proximity, rendering only occurs for entities within the client’s field-of-view. Furthermore, the transparency of entities is a function of the entities’ distance from the user’s current location. For the line strips that define streets/roads and building footprints, the transparency is determined with a linearly decreasing function with increasing distance away from the user. For annotations, the transparency is also a linear function but is combined with a step function to only apply fading to annotations within a certain distance.

4.4.4 User Interface and Experience

One of the features that was implemented to improve the user experience was to render the traditional overhead view of the map surrounding a user’s position when the user aimed the iPhone towards the ground. This was done because the position that the user holds the iPhone in was reminiscent of typical usage of modern mobile devices: users will look downwards at their mobile device to view what is being displayed on the screen. To perform the transition to the birds-eye view, the accelerometer pitch-angle is used. The transition is performed gradually, as a linear function of the difference between the tilt-angle and the target pitch angle of -90 degrees. The transparency of the video frame rendered in the background also uses the same linear function.
Another feature that was implemented stemmed from the birds-eye view: in order for a user to view which direction they are facing, a compass is rendered on the floor of the virtual environment where the user is standing. The compass and the birds-eye view are illustrated in Figure 4-5 below:

**Figure 4-5: Birds-eye view of virtual environment with compass rendering.**
The above figure is a rendering of the virtual environment at the King’s College Road and Circle in Toronto, Canada from an overhead (i.e., birds-eye) perspective. The user’s position is indicated by the center of the compass near the center of the figure. The camera’s field of view and viewing direction are indicated by the triangular shaded region extending outwards from the center of the compass towards the top of the figure.

Another feature that was implemented was to highlight the building that the user was currently viewing. To do this, the building annotation and associated arrow bounce up and down to catch the user’s attention as to what building is currently highlighted. This is illustrated in Figure 4-6 below:
Figure 4-6: Highlighting of building within user’s field-of-view.
The above figure is a rendering of the virtual environment in front of Convocation Hall at King’s College Road in Toronto, Canada. The video frame that would normally be rendered in the background is removed for clarity. The building footprint of Convocation Hall is extruded vertically to place emphasis on it. The building annotation and arrow marker are animated to bounce up and down to emphasize that the building is highlighted.

Example photographs of the client running on an iPhone 3G are shown in Figure 4-7:

Figure 4-7: Client running on an iPhone 3G.
The photos above illustrate the client running the navigational application on an iPhone 3G. The photos were taken on Dundas Street West in Toronto, Ontario across from the Art Gallery of Ontario. In all of the photos, Dundas Street West is rendered as a transparent yellow line overlaid on top of the actual street seen from the iPhone’s camera. The actual name of the street is rendered as floating text above the yellow line and is oriented such that the text is aligned with the street.
4.5 Chapter Summary

This chapter provided an overview of the system that was designed and implemented. In particular, the source server, the main server, and client component implementations were discussed in detail. Chapter 5 will provide evaluation results for the system.
5 Evaluation and Results

5.1 Overview

To evaluate the system, the computational run-time and the accuracy of the camera yaw estimation approach is investigated. This is because the camera yaw estimation is the largest source of error due to the approach using relative frames of reference. It also consumes the most computational resources due to the image processing required for feature extraction. The camera position and pitch angle determination approaches on the other hand, use absolute frames of reference and are insignificant in terms of computational run-time.

In addition, the viability for mainstream use is investigated by evaluating the overall performance of the navigational application. In particular, the overall rendering speed and usability of the application is investigated informally at a high-level.

The testing methodology used to evaluate the system and the results obtained are described in the following two sections.

5.2 Methodology

To evaluate the performance and accuracy of the yaw estimation approach in a repeatable and consistent manner, test data consisting of video captured from a mobile device with a digital compass was first acquired. This would allow for different test configurations to be run on the same data set to evaluate their relative performance and accuracy.

The test data was acquired using a test setup consisting of an Ocean Server Technology OS5000-US digital compass, a netbook PC, a Nokia N82 camera phone, and a Hauppauge WinTV2. The compass was mounted on top of the N82 and connected to the netbook via a USB
The compass was configured to use a baud rate of 115,200 bps and an output rate of 40 samples per second. The N82 was configured to output an NTSC video signal to the WinTV2, which was in turn connected to the netbook via a USB cable. A data logging application was developed to log the timestamped compass samples into an SQLite3 database and to encode the video frames captured from the WinTV2 into a 640x480 MPEG-4 video with 3 mbits/sec bitrate and variable frames per second using V4L2 and the ffmpeg library (libavutil 49.10.0, libavcodec 51.71.0, libavformat 52.22.1, and libavdevice 52.1.0). The test setup is illustrated in Fig. below:

Figure 5-1: Test setup used for data acquisition.
The above figure is an abstract illustration showing the devices used for test data acquisition and their connectivity. The digital compass is mounted on the Nokia N82 and is held by the user. The N82 is connected to a WinTV2 capture device using its NTSC video-out connector and a composite NTSC video cable. Both the digital compass and the WinTV2 capture device are connected to a netbook personal computer via USB cables. A data logging application is executed to read and parse compass samples as well as capture video frames obtained from the WinTV2 device using V4L2. The compass samples are timestamped and stored to a database using the SQLite3 database engine. The video frames are encoded into a variable fps (vfps) MPEG-4 video file.

Data was captured for two locations at the University of Toronto near King’s College Circle with GPS coordinates $LOC1 = (43.66056151, -79.39491436)$ and $LOC2 = (43.66140267, -79.39500689)$. The test setup was placed at those locations with one user holding the N82/compass to aim it around to view the surrounding environment and another user to execute
and terminate the data logging application. After each capture session, the data would be annotated to describe the actions and scene conditions. These are described in Table 5-1 below:

Table 5-1: Test Data Descriptions

<table>
<thead>
<tr>
<th>LOC1: Meridian Plaque Outside Sanford Fleming Building</th>
<th>Description of Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test ID</td>
<td>Description</td>
</tr>
<tr>
<td>1A</td>
<td>This video shows the user aiming the camera left and right in a slow, continuous motion at the beginning. At the end of the video, the user rotates the camera quickly. This video contains some slight fluctuating gain, occluding objects (a moving vehicle, a person walking in the distance), and interlacing artifacts introduced from the WinTV2 capture device.</td>
</tr>
<tr>
<td>1B</td>
<td>The video is similar to 1A, however there are many more occluding objects, namely many people walking. This video is used to illustrate performance under a more realistic scenario.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LOC2: Southern End of Field at King’s College Circle</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test ID</td>
<td>Description</td>
</tr>
<tr>
<td>2A</td>
<td>This video shows the user aiming the camera moderately slowly to the right until a full revolution is performed. The user then aims the camera left rapidly. This video contains some slight fluctuating gain due to the glare of the sun, no significant occluding objects, and interlacing artifacts.</td>
</tr>
<tr>
<td>2B</td>
<td>This video shows the user aiming the camera continuously towards the right, making over two complete revolutions. This video contains some occluding objects, fluctuating gain due to the glare of the sun, interlacing, and low framerate. This video is used to investigate drift error under continuous motion and rotation speed.</td>
</tr>
</tbody>
</table>

Once the data had been acquired, the yaw estimation approach was applied using different configurations to evaluate the relative performance and accuracy improvements or degradations. The evaluation was performed on a desktop workstation with an Intel Core 2 Quad Q6600 2.4GHz using a single-threaded implementation of the navigational application built using GCC 4.3.0 with the following compilation flags: `-O5 -fno-strict-aliasing -ffast-math -fomit-frame-pointer`. The navigation application was modified to use the video files and compass database as input. In particular, the initial camera heading used was the average of the first 30 compass samples. Furthermore, the timestamps of the video frames were used to
index into the compass database to determine which compass heading samples to compare the calculated yaw angles with. The navigation application was also configured to use a camera field-of-view of 55 degrees for the N82, a resolution of 640x480 to match the input video resolution, an estimated distance of 0.1 m between the camera and the user position, and to use the median for horizontal displacement determination.

During each evaluation run, the estimated yaw angle calculated was logged as well as timing information for relevant sections of the code. To benchmark the different sections of code, `gettimeofday()` was used to accumulate the total execution times and counter variables were used to keep track of how many times the sections of code were executed. Average execution times are then calculated as a post-process by dividing the total execution time with the number of times the code was executed.

Using the results from the evaluation, a tracking configuration suitable for real-time execution on the iPhone 3G was chosen. A high-level quantitative and qualitative usability evaluation was then performed with the iPhone 3G using the navigation application and the chosen configuration. The results from both evaluations are discussed in the following section.

### 5.3 Results

The results for the yaw angle estimation in comparison to the digital compass heading are presented as angle graphs. The angle graphs show the general behaviour of the yaw angle estimation approach versus the actual compass heading. The drift error that is accumulated can be determined by integrating the difference of the angle graphs and the actual compass heading.
The computational run-times of the SURF feature extraction and matching routines are also shown as they are the primary indicators of the performance of the yaw estimation approach as it was empirically determined that they occupy the most execution time.

Real-time performance was not obtained using 640x480 resolution images using a 2.4 GHz processor. To match the performance of what would be seen using the iPhone 3G’s resolution (320x480), image down-sampling and specifying a region of interest is investigated.

5.3.1 Image Down-sampling

Down-sampling of the image buffers was performed using OpenCV’s `cvResize` function. The original image buffer was down-sampled by factors of 2 and 4. The results are presented below with $DS2$ denoting down-sampling by a factor of 2 (i.e., 320x240 resolution) and $DS4$ denoting down-sampling by a factor of 4 (i.e., 160x120 resolution).

![Graph showing yaw angle estimation with image down-sampling for Test 1A.](image-url)

Figure 5-2: Yaw angle estimation with image down-sampling for Test 1A.
The results for Test 1A in Figure 5-2 above show that DS2 and DS4 both still exhibit behaviour similar to the actual compass heading. However, the total drift error increases by 110.86% when going from DS2 to DS4. The average number of feature correspondences per frame pair with DS2 was found to be 194.44, whereas with DS2 the average drops to 65.51 (a 66.3% decrease). The estimated heading with DS2 from beyond the 45 second mark drifts further than with DS4 due to the quicker camera rotation: with DS2, more features were able to be matched during the accelerated motion which yielded a higher horizontal displacement on average than with DS4.

![Test 1B Graph](image)

**Figure 5-3: Yaw angle estimation with image down-sampling for Test 1B.**

The results for Test 1B in Figure 5-3 above show very similar behaviour for DS2 and DS4. The average number of feature correspondences per frame pair with DS2 was found to be 243.53 versus 76.96 with DS4 (a 68.4% decrease). The drift error however was lower for DS4 than with DS2 (approximately 9.78% less drift). This is because in Test 1B, there are many
moving objects and DS2’s yaw estimation is slightly affected due to the higher number of features matched.

![Test 2A](image)

**Figure 5-4: Yaw angle estimation with image down-sampling for Test 2A.**

The results for Test 2B in Figure 5-4 above show the yaw estimation with DS4 performing better than with DS2. The drift error observed with DS4 is 28.77% less than with DS2, even though the average number of feature correspondences per frame pair with DS4 (56.02) is 69.56% less than the average number of feature correspondences with DS2 (184.015). The reason for this is the higher on average horizontal displacement calculated with DS2 during the initial revolution in Test 2B. This is most likely due to the greater number of features correspondences skewing the median, which resulted from the higher resolution and slow camera motion. Another possibility could be that the spatial distribution of the feature correspondences is greater with DS4 than with DS2, which could result in better horizontal displacement determination on average. This is because with lower down-sampling factors, clusters of correspondences (i.e., correspondences that are in close proximity to each other in image space)
which have similar horizontal displacements are a more common occurrence and as a result, the median may be skewed.

![Figure 5-5: Yaw angle estimation with image down-sampling for Test 2B.](image)

The results from Test 2B in **Figure 5-5** show that again, the estimated yaw angle behaves very similar to the actual compass heading. The accuracy near the end is ignored due to the test video containing a static frame from the 47 second mark onwards. With DS4, the drift error is 46.68% less than with DS2. The average number of feature correspondences is 65.19% less with DS4 in comparison to DS2. DS4 performs better than DS2 for similar reasons as with Test 2A. Also, video artifacts may be more visible with DS2 than with DS4 and as such, affect the feature correspondences used for estimated the heading.

The average execution times for the SURF feature extraction, feature matching, and the entire frame processing for yaw angle estimation across all of the tests are listed in **Table 5-2** below:
Table 5-2: Average execution times with image down-sampling.

<table>
<thead>
<tr>
<th>Test</th>
<th>Feature Extraction (ms)</th>
<th>Feature Matching (ms)</th>
<th>Process Frame (ms)</th>
<th>Average Number of Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Down-sampling by Factor of 2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1A</td>
<td>55.59</td>
<td>38.54</td>
<td>96.04</td>
<td>194.44</td>
</tr>
<tr>
<td>1B</td>
<td>57.82</td>
<td>38.13</td>
<td>97.94</td>
<td>243.53</td>
</tr>
<tr>
<td>2A</td>
<td>49.34</td>
<td>29.67</td>
<td>80.88</td>
<td>184.02</td>
</tr>
<tr>
<td>2B</td>
<td>37.08</td>
<td>18.01</td>
<td>56.99</td>
<td>142.39</td>
</tr>
<tr>
<td><strong>Avg</strong></td>
<td><strong>49.96</strong></td>
<td><strong>31.09</strong></td>
<td><strong>82.96</strong></td>
<td><strong>191.10</strong></td>
</tr>
<tr>
<td><strong>Down-sampling by Factor of 4</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1A</td>
<td>11.83</td>
<td>2.69</td>
<td>15.83</td>
<td>65.51</td>
</tr>
<tr>
<td>1B</td>
<td>11.62</td>
<td>2.50</td>
<td>15.47</td>
<td>76.96</td>
</tr>
<tr>
<td>2A</td>
<td>11.34</td>
<td>2.09</td>
<td>14.91</td>
<td>56.02</td>
</tr>
<tr>
<td>2B</td>
<td>9.66</td>
<td>1.51</td>
<td>12.61</td>
<td>49.57</td>
</tr>
<tr>
<td><strong>Avg</strong></td>
<td><strong>11.11</strong></td>
<td><strong>2.20</strong></td>
<td><strong>14.71</strong></td>
<td><strong>62.01</strong></td>
</tr>
</tbody>
</table>

From the average execution times listed in Table 5-2 above, down-sampling by a factor of 4 versus a factor of 2 resulted in speedups in execution time by factors of 4.5, 14.16, and 5.64 for feature extraction, matching, and frame processing respectively. Since the tests were performed on a 2.4 GHz machine, down-sampling by a factor of 2 would not yield real-time performance on the iPhone 3G’s 412 MHz ARM processor. Furthermore, since down-sampling by a factor of 4 yielded similar or better accuracy than down-sampling by a factor of 2 from the results mentioned previously, the most appropriate image resolution for use on the iPhone 3G would be approximately 160x120. This implies that a down-sampling factor of at least 2 would be required since the iPhone 3G’s image resolution is 320x480.

### 5.3.2 Region of Interest

Instead of down-sampling the source video frames, to reduce the number of pixels scanned to achieve real-time performance, the region of interest that the SURF algorithm operates is adjusted to be 25% of the height of the original video frame with the same width (i.e., 640x120). This yields the same number of pixels to operate on as with DS2 (320x240). For this reason, evaluation will be performed in comparison to using DS2.
The region of interest is positioned at the center of the original video frame for this evaluation. This is because it is estimated to have the most scene content as users generally aim the camera such that objects of interest are in the center of the field of view. Therefore, the center region could have a higher likelihood of having unique features to extract. The results from the evaluation are shown below.

![Graph](image)

**Figure 5-6: Yaw angle estimation with region of interest for Test 1A.**

The results from Test 1A in Figure 5-6 above show very similar performance to the results when down-sampling is performed. The drift error is within 0.5% when compared with DS2. However, a higher drift is observed during the final rotation during the test where the camera is rotated rapidly. The average number of feature correspondences is 171.8 which is less than with DS2 (194.44). This is most likely due to fact that the region of interest does not consider the top and bottom edges of the image buffer. These results reveal that during the rapid camera motion, considering the entire image buffer yielded reduced drift error.
Figure 5-7: Yaw angle estimation with region of interest for Test 1B.

The results for Test 1B in Figure 5-7 above show worse performance than with DS2. The drift error was found to be 18.2% higher than with DS2. The average number of feature correspondences (167.58) was 31.2% less than with DS2 (243.53) due to similar reasoning as the results for Test 1A. The initially large drift error at the beginning of the test is due to the many moving objects within the region of interest moving in the opposite direction of the camera rotation. This resulted in a higher median horizontal displacement which indicates that this is an instance where considering the entire frame would have yielded a better estimation. In other words, considering the entire image buffer would have been more robust to occluding objects.

The results for Test 2A in Figure 5-8 below show similar performance to using DS2. The drift error was slightly less than DS2 (5.5%). The similar performance is most likely due to the static scene content of the test. The results for Test 2B in Figure 5-9 below show higher drift...
error than with DS2. This is due to features experiencing higher horizontal displacement near the center of the image buffer than features near the bottom during camera rotation.

Figure 5-8: Yaw angle estimation with region of interest for Test 2A.

Figure 5-9: Yaw angle estimation with region of interest for Test 2B.
The average execution times for the SURF feature extraction, feature matching, and the entire frame processing for yaw angle estimation across all of the tests are listed in Table 5-3 below:

**Table 5-3: Average execution times with specifying a region of interest.**

<table>
<thead>
<tr>
<th>Test</th>
<th>Feature Extraction (ms)</th>
<th>Feature Matching (ms)</th>
<th>Process Frame (ms)</th>
<th>Average Number of Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1A</td>
<td>50.85</td>
<td>53.44</td>
<td>105.86</td>
<td>171.81</td>
</tr>
<tr>
<td>1B</td>
<td>38.44</td>
<td>20.17</td>
<td>60.24</td>
<td>167.58</td>
</tr>
<tr>
<td>2A</td>
<td>31.67</td>
<td>14.58</td>
<td>47.81</td>
<td>100.21</td>
</tr>
<tr>
<td>2B</td>
<td>28.39</td>
<td>10.36</td>
<td>40.74</td>
<td>64.94</td>
</tr>
<tr>
<td>Avg</td>
<td>37.34</td>
<td>24.64</td>
<td>63.66</td>
<td>126.14</td>
</tr>
</tbody>
</table>

It is observed that down-sampling by a factor of 2 performs 1.2 times slower than when specifying a region of interest occupying the same amount of pixels. This is mostly due to the increased number of feature correspondences on average when down-sampling.

From the results of evaluating the estimation approach when using a region of interest, similar performance to down-sampling by a factor of 2 is observed. Down-sampling however, experiences less drift during faster camera rotations, scenes with moving objects, and scenes with static scene content.

### 5.4 Usability

Based on the results in Section 5.3, down-sampling by a factor of 2 was used for the yaw estimation approach on the iPhone 3G. This allowed for an overall rendering rate of approximately 12-15 Hz when running the navigational application. This provided adequate smoothness when updating the camera orientation based on informal feedback from several users. The downloading and loading of map tiles took approximately 10-20 seconds depending
on the amount of data within the tiles. This was found to be acceptable since the download would only be performed once per tile and the delay would only be noticeable at application startup. Subsequent downloading and loading of tiles would be performed in the background and was not noticeable. However, with the yaw estimation approach executing, the load times of these subsequent tiles was found to have approximately doubled.

The positional update interval from the GPS receiver fluctuated significantly at times, ranging from 1 second to over a minute. Since the initial heading estimate used for the yaw estimation approach is derived from the GPS receiver, large update intervals caused the application to become completely inaccurate and unusable. Furthermore, having to both walk continuously to obtain a more accurate GPS heading and initially aim the iPhone in the walking direction proved to be restricting and cumbersome.

The battery life of the iPhone 3G when running the navigational application continuously was found to be less than one hour. This suggests that the application should be used on occasion for brief periods of time to allow a user to figure out their surroundings and not to continuously explore the environment as one would with a head-mounted display.

While using the navigational application, it was observed that the passersby would generally not pay much attention or are indifferent to seeing a user aim an iPhone around the environment, as it has become more commonplace to see people take pictures using their mobile devices. Also, while using the navigational application, it was observed that rendering the building footprints and road geometry would actually accentuate the drift error from the tracking estimation approach. Similarly, the compass rendered on the ground of the virtual environment would also accentuate the tracking error if one was familiar with the actual relative headings of
specific landmarks in the environment. For this reason, only annotations should be rendered as they still remained accurate during light usage (i.e., when undergoing small rotations).

5.5 Chapter Summary

This chapter summarized the performance and accuracy results of the yaw angle estimation approach using different configurations. From the results, a suitable configuration was chosen for execution on the iPhone 3G. In terms of high-level usability, the navigation application was found to be reasonably accurate but due to drift error, a user can only aim it around the environment for a relative short period of time, especially if the user’s motions undergo large rotational displacements. The perceived accuracy was found to be addressable by not rendering the building footprints or the compass and just rendering the floating annotations.

The following sections will draw conclusions from the findings in this dissertation as well as discuss future work and potential applications.
6 Conclusion

This dissertation presented findings in developing a real-time, mediated reality platform for the iPhone 3G that can be extended to other mobile devices. In particular, an outdoor navigational application was developed using the platform to provide information about a user’s surroundings in the form of text or graphics overlaid on top of what a user sees. A hybrid orientation tracking approach using GPS and accelerometer sensors as well as computer vision was also developed and evaluated. Furthermore, the server infrastructure to support the navigational application for use in different geographical regions throughout the world was also developed. Lastly, a method for dynamically downloading and loading different regions of the world in a seamless manner was implemented.

It was found that the tracking approach was capable of per-pixel tracking accuracy under specific conditions. Under general conditions however, the tracking approach suffers from drift error and provides usable accuracy only when the camera undergoes small rotations (i.e., less than one revolution).

The computational run-time of the tracking approach was found to be largely dictated by the yaw estimation approach due to the use of SURF. It was found that down-sampling the reference images yielded less drift error than when specifying a region of interest during faster camera rotations, scenes with moving objects, and scenes with static scene content. Down-sampling also reduced the run-time of the feature extraction process, which allowed for real-time performance on the iPhone 3G.

From informal usability findings, it was found that having to be constrained to certain usage patterns for correct operation was too cumbersome for users. Also, it was found that the
perceived accuracy of the tracking approach could be improved by only rendering annotations and not the virtual environment geometry such as streets or building footprints.

Although this research has made some considerable first steps in bringing mediated reality to the mainstream public, there still remains future work to allow for this research to be truly viable for mainstream usage and adoption. The future work and other potential applications are discussed in the next section.
7 Future Work

7.1 Improvements

Although the prototype system developed is functional, there are potential improvements that can be made in terms of computational performance (i.e., response time and rendering latency) as well as orientation tracking accuracy. For instance, the use of reference features may potentially reduce the drift error and place a limit on the maximum accumulated error that is observed. The premise would be to build a lookup-table of features and their estimated headings as they are extracted. If they are detected again at a later time, the previously estimated heading for the feature(s) can be used instead of performing yaw estimation again and accumulating more error. This would be especially useful for situations such as when the camera undergoes full revolutions: when the user completes one revolution, some features that would be extracted at that point in time would ideally be found in the lookup-table and the previously estimated headings could be used. This however, has the potential to be memory-intensive as the number of features that need to be stored in the lookup-table can be quite large, depending on the scene content.

However, with the advent of mobile phones (such as the iPhone 3GS) with increased computational resources and integrated inertial sensors such as digital compasses, these improvements may find themselves to be useful only on other platforms or in other applications.

7.2 Potential Applications

The prototype system developed can also be leveraged as a framework or platform for a variety of applications such as navigational aids, social networking, and personal safety.
For navigation applications, the platform can be extended to overlay directions onto roads for a user to reach a destination, similar to commercially available GPS navigation systems. Furthermore, if other contextually-related information was available, the platform could be extended to provide that information as a user navigates their environment (e.g., to act as a tour-guide).

This platform can also be extended to support the addition of user-generated content such as messages: users could leave virtual messages to annotate the environment for other users to see. For example, a user could leave a message to another group of users in front of a building to indicate where all of the users should meet. For users who do not know the area that well, they would simply have to aim their mobile device around to see where the virtual message is floating to navigate to where they should go.

This platform could also help determine a user’s context for applications such as automated content tagging. For example, if a user is taking a photograph of a given landmark, the user’s location, orientation, and SURF features extracted from the photograph could be used to more efficiently search through a database populated with known landmarks in order to identify the content of the photograph automatically.

If other sensory information could be acquired and captured, this platform could be extended to perform real-time analysis to more accurately determine a user’s context. If this information were to be logged, then the platform could in essence serve as a personal safety device in case of emergency situations. The information could also be used for summarizing the daily events in a person’s life if they used it on a daily basis.
References


[31] M. Kahari and D.J. Murphy, “MARA, sensor based augmented reality system for mobile imaging device,” Proc. of Fifth IEEE ISMAR.


at:


[49] “API v0.6 - OpenStreetMap.” Available at: http://wiki.openstreetmap.org/wiki/API_v0.6


Appendices

Appendix A: Example OpenStreetMaps XML Data

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="OpenStreetMap server">
  <bounds minlat="43.6594" minlon="-79.39721" maxlat="43.66092" maxlon="-79.39281"/>
...
  <node id="24960080" lat="43.6610685" lon="-79.3948749" version="7" changeset="871262" user="s021" uid="112325" visible="true" timestamp="2009-03-30T22:36:50Z"/>
  <node id="123347786" lat="43.6599032" lon="-79.3943929" version="6" changeset="91097" user="andrewpmk" uid="1679" visible="true" timestamp="2009-02-03T08:58:05Z"/>
  <node id="80927373" lat="43.6590906" lon="-79.3940428" version="7" changeset="91097" user="andrewpmk" uid="1679" visible="true" timestamp="2009-02-03T09:26:08Z"/>
...
  <way id="4675080" visible="true" timestamp="2009-04-04T04:09:02Z" version="8" changeset="169582" user="andrewpmk" uid="1679">
    <nd ref="24960080"/>
    <nd ref="123347786"/>
    <nd ref="80927373"/>
    <tag k="name" v="King's College Road"/>
    <tag k="lit" v="yes"/>
    <tag k="surface" v="cobblestone"/>
    <tag k="maxspeed" v="20"/>
    <tag k="created_by" v="Potlatch 0.10d"/>
    <tag k="operator" v="University of Toronto"/>
    <tag k="highway" v="residential"/>
    <tag k="access" v="permissive"/>
  </way>
...
</osm>
```

The listing above depicts an excerpt from the XML data retrieved from a query to the OpenStreetMaps server using API v0.6 for entities within the bounding box defined by the coordinates of the XML `bounds` node:

```
http://api.openstreetmap.org/api/0.6/map?bbox=left,bottom,right,top
```
(Where left, bottom, right, and top are GPS coordinates defining the bounding box)

A particular `way` is shown, defining a portion of King’s College Road in Toronto, Ontario by referencing three nodes (node id 24960080, 123347786, and 80927373). The three nodes are defined earlier on in the XML data.

Tags are associated with the way to provide extra information describing the way. For example, the `highway` tag specifies that the way represents a residential road.
Appendix B: SQLite Geo-Database Structure

```sql
PRAGMA encoding = "UTF-8";
CREATE TABLE nodes (  id INTEGER PRIMARY KEY,  longitude REAL,  latitude REAL,  version INTEGER,  timestamp REAL,  user TEXT);
CREATE TABLE node_tags (  id INTEGER,  key TEXT,  value TEXT);
CREATE TABLE ways (  id INTEGER,  version INTEGER,  timestamp REAL,  user TEXT);
CREATE TABLE way_refs (  id INTEGER,  num INTEGER,  node INTEGER);
CREATE TABLE way_tags (  id INTEGER,  key TEXT,  value TEXT);
```

The listing above is the SQL script that is used to create the SQLite3 databases for storing the geo-data.

Nodes and ways are stored in their own tables and contain versioning information to allow for updates to occur in the future if they are modified.

Node and way tags are separated from the nodes and ways due to the arbitrary number of tags allowed for each individual node or way.

The way.refs table stores the node references that define the way in sequential order as indicated by the ‘num’ column.
Appendix C: iPhone 3G Raw Camera Frame Acquisition

The iPhone 3G OS version 2.2.1 does not provide official API support for acquiring raw camera frames. However, workarounds exist that allow for this functionality. Example source code can be found at:

http://github.com/norio-nomura/iphonetest/tree/9713242dda6c6bc897da4bd639a1fdadc29b6fd7/CameraTest

The classes implement surfaces for capturing and transferring the bitmap data from the camera as well as a callback to copy the pixel data from the surface into a usable 2D byte array for image processing. The functions to install the camera callback and the actual callback function from the above example source are shown in the code listing below:

```c
#import <objc/runtime.h>
#import "CameraTestAppDelegate.h"
#import "Surface.h"
#import "SurfaceAccelerator.h"

OBJC_EXPORT unsigned int CGBitmapGetFastestAlignment();
OBJC_EXPORT void * CGBitmapAllocateData(unsigned int);
OBJC_EXPORT void CGBitmapFreeData(const void *data);

static FUNC _camera_callback original_camera_callback = NULL;
static void *readblePixels = NULL;

static int __camera_callbackHook(
    CameraDeviceRef cameraDevice,
    int a,
    CoreSurfaceBufferRef coreSurfaceBuffer,
    int b
) {
    CoreSurfaceAcceleratorRef coreSurfaceAccelerator = *(CoreSurfaceAcceleratorRef*)(cameraDevice+84);
    unsigned int surfaceId = [Surface CoreSurfaceBufferGetID:coreSurfaceBuffer];
    if (coreSurfaceBuffer) {
        Surface *surface = [[Surface alloc]
            initWithFrame:coreSurfaceBuffer];
        [surface lock];
        unsigned int height = surface.height;
        unsigned int width = surface.width;
        unsigned int alignment = CGBitmapGetFastestAlignment();
        unsigned int alignmentedBytesPerRow = (width * 4 / alignment + 1) * alignment;
        if (!readblePixels) {
            readablePixels = CGBitmapAllocateData(alignmentedBytesPerRow * height);
        }
        unsigned int bytesPerRow = surface.bytesPerRow;
        void *pixels = surface.baseAddress;
        for (unsigned int j = 0; j < height; j++) {
            for (unsigned int i = 0; i < bytesPerRow; i++) {
                // Copy pixel data
            }
        }
    }
    return 0;
}
```
memcpy(
    readablePixels + alignmentedBytesPerRow * j, pixels +
    bytesPerRow * j,
    bytesPerRow
);
} [surface unlock];
} [surface release];
return (*original_camera_callback)(cameraDevice,a,coreSurfaceBuffer,b);

...  

@implementation CameraTestAppDelegate

@synthesize window;
@synthesize cameraController;
...

- (void)install_camera_callbackHook {
    char *p = NULL;
    object_getInstanceVariable(cameraController,"_camera",(void**)&p);
    if (!p) return;
    if (!original_camera_callback) {
        FUNC_camera_callback *funcP = (FUNC_camera_callback*)p;
        original_camera_callback = *(funcP+37);
        (funcP+37)[0] = __camera_callbackHook;
    }
}
...

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    application.statusBarHidden = YES;
    self.cameraController = [objc_getClass("PLCameraController")
        sharedInstance];
    [cameraController setDelegate:self];
    UIView *previewView = [cameraController previewView];
    [cameraController startPreview];
    [Surface dynamicLoad];
    [self install_camera_callbackHook];
    [window addSubview:previewView];
    [window makeKeyAndVisible];
}
...
@end