A MARKERLESS AUGMENTED REALITY SYSTEM FOR MOBILE DEVICES

By
Alex Ufkes
Bachelor of Science
in the Program of Computer Science,
Ryerson University, 2010

A thesis
presented to Ryerson University
in partial fulfilment of the
requirements for the degree of
Master of Science
in the Program of
Computer Science

Toronto, Ontario, Canada, 2013

ABSTRACT

A MARKERLESS AUGMENTED REALITY SYSTEM FOR MOBILE DEVICES

Alex Ufkes

Master of Science, Computer Science, Ryerson University, 2013

Augmented Reality (AR) combines a live camera view of a real world environment with computer-generated virtual content. Alignment of these viewpoints is done by recognizing artificial fiducial markers, or, more recently, natural features already present in the environment. This is known as Marker-based and Markerless AR respectively.

We present a markerless AR system that is not limited to artificial markers, but is capable of rendering augmentations over user-selected textured surfaces, or ‘maps’. The system stores and differentiates between multiple maps, all created online. Once recognized, maps are tracked using a hybrid algorithm based on feature matching and inlier tracking.

With the increasing ubiquity and capability of mobile devices, we believe it is possible to perform robust, markerless AR on current generation tablets and smartphones. The proposed system is shown to operate in real-time on mobile devices, and generate robust augmentations under a wide range of map compositions and viewing conditions.

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr. Mark Fiala for his direction and support, both technically and financially. His guidance made this thesis possible and inspired me to pursue research in the field of computer vision. I look forward to continued collaboration with him as I continue my graduate career.

I would also like to thank my co-supervisor Dr. Alex Ferworn for encouraging me to join the graduate program at Ryerson and allowing me to participate in his research. He has inspired me to pursue a Ph.D. in computer science, and I look forward to studying under his supervision.

Finally I would like to thank everyone in the N-CART lab who made pursuing a Master’s degree more enjoyable than I ever thought possible. Specifically I would like to thank Jimmy Tran, who has been a constant source of advice and knowledge throughout my time at Ryerson.

This work was funded by NSERC Discovery Grant 356072 and the Faculty of Engineering and Architectural Science at Ryerson University.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF APPENDICES

ABBREVIATIONS

AR	Augmented Reality
BoW	Bag of Words
BRIEF	Binary Robust Independent Elementary Features
CPU	Central Processing Unit
EPnP	Efficient Perspective n-Point
FAST	Features from Accelerated Segment Test
GPS	Global Positional System
GPU	Graphics Processing Unit
HMD	Head-Mounted Display
KLT	Kanade-Lucas-Tomasi
NCC	Normalized Cross Correlation
ORB	Oriented BRIEF
PDA	Personal Digital Assistant
PnP	Perspective n-Point
PTAM	Parallel Tracking and Mapping
RANSAC	RANdom SAmple and Consensus
RGB	Red-Green-Blue colour space
RoI	Region of Interest
SAD	Sum of Absolute Differences
SIFT	Scale-Invariant Feature Transform
SIMD	Single Instruction Multiple Data
SLAM	Simultaneous Localization and Mapping
SSD	Sum of Squared Differences
SURF	Speeded-Up Robust Features
VR	Virtual Reality
YUV	Colour space based on luminance and chrominance

CHAPTER 1 INTRODUCTION

1.1 Introduction

Augmented Reality (AR) is a sub-class of Virtual Reality (VR). In true virtual reality applications, all sensory input experienced by the user is simulated. In augmented reality, virtual content is combined with real world sensory input to create a blended version of the environment. Early attempts at AR utilized orientation and Global Positioning System (GPS) sensors to align the overlaid graphics. Due to the limited accuracy of these sensors, the results were poor. Modern approaches use computer vision techniques to analyze live video input from a hand-held camera or Head-Mounted Display (HMD) to determine the location and orientation (pose) of the camera in 3D space. Virtual augmentations can then be inserted seamlessly into the real-world video feed using this pose information.

The appeal of virtual reality is that it puts complete control of the environment in the hands of users or operators. However, even modern technology cannot produce completely realistic VR, and therefore the user's sense of immersion is not total. While AR systems do not improve on the virtual content itself, by placing it in the context of the real world environment a heightened sense of immersion and believability in the simulation can be achieved. Even when immersion is not a requirement, AR offers a means of delivering virtual content without removing the user from their environment.

Early applications of AR identified by Woodwark in 1993 [1] included aiding maintenance workers; a virtual, labelled car engine could be aligned with the real engine, showing workers what to adjust or fix, or what to do next in an assembly line situation (Figure 1.1). Another application is Computer Aided Design (CAD). Synthetic images of future structures have been placed over landscapes for planning purposes as early as 1973

[2], the logical extension of which is to augment entire virtual 3D structures over potential building sites. More recent applications of AR include gaming [3, 4], aiding medical professionals [5, 6], head-up displays in luxury cars [7], and intuitively viewing and manipulating 3D data [8].

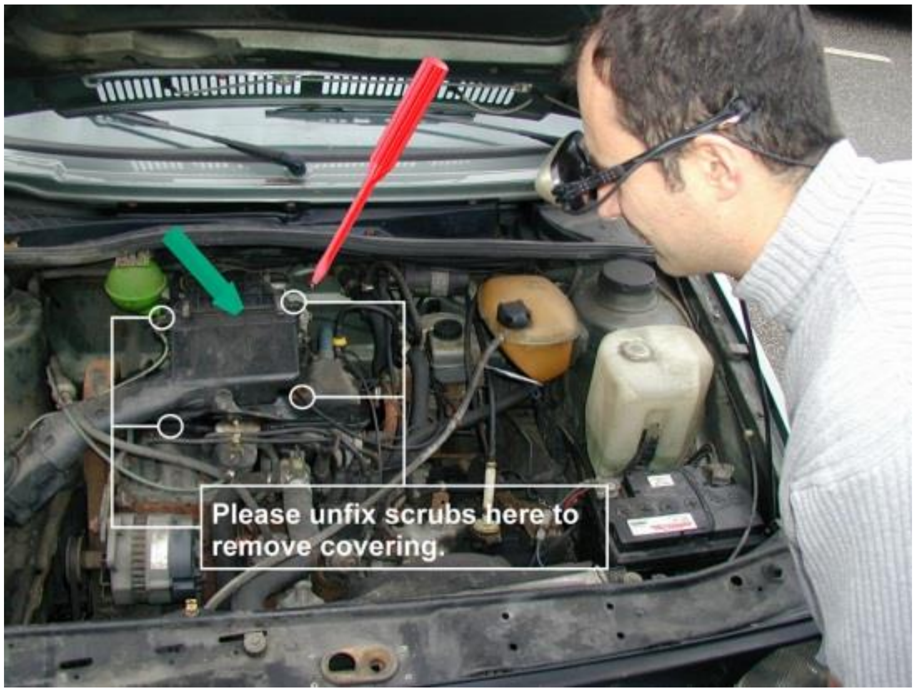

Figure 1.1 - Assembly instructions are inserted into the user's field of view via a head mounted display (image and caption from [9]).

1.1.1 Marker-based AR

Traditional AR systems such as ARTag [10], the AR-Toolkit project [11], and Nintendo's 3DS handheld gaming platform [4] utilize fiducial markers or "tags", over which virtual objects are rendered (Figure 1.2).

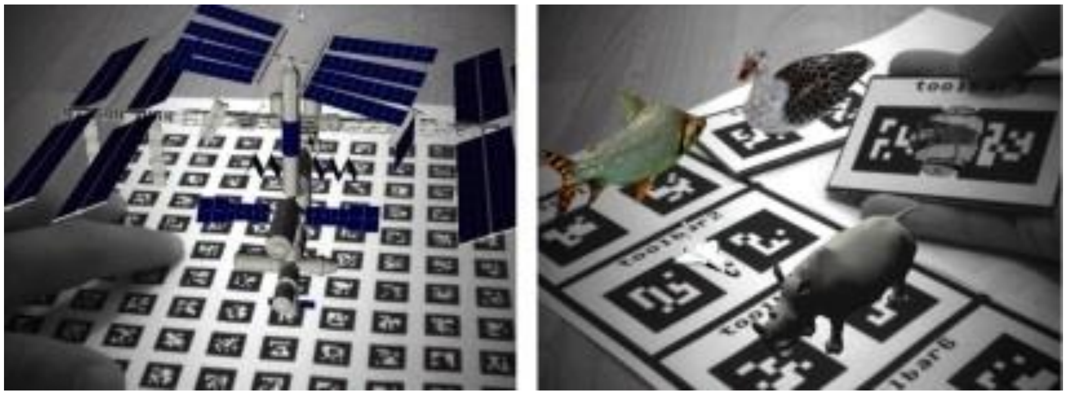

Figure 1.2 - Virtual objects rendered over markers using the ARTag system from [10].

These tags are similar to bar codes or QR codes. They are unique and visually distinct, making them robustly identifiable. When a marker is located in the camera frame, its image coordinates are used to compute the camera pose and draw augmentations. Marker-based systems are ideal for applications involving a static or fixed environment, or situations where the desired virtual space does not extend beyond the tag itself. For more ubiquitous and ad-hoc AR applications, markerless AR has emerged.

1.1.2 Markerless AR

Instead of relying on artificial markers for determining camera pose, markerless augmented reality systems make use of natural features already present in the environment. The workflow to determine camera pose using natural features is more complex than marker-based systems, involving additional algorithms to find and match these natural features between camera frames. Once features are matched or tracked, camera pose is calculated from these correspondences. By not relying on markers, these systems can be utilized ad-hoc in almost any situation or location. There is no need to modify environments ahead of time, though it is common to require that a map or log of the natural features within an environment be created beforehand, as in [12].

1.2 Problem Definition

The main limitation of traditional marker-based AR is the markers themselves. In the case of simple applications, users must print and carry the tags (or in some cases spend time attaching them to the environment) to use the AR application [13]. For more complex, large scale applications, significant infrastructure is required in the form of dozens or hundreds of markers placed throughout the environment. These requirements severely inhibit the implementation of ubiquitous AR. It is not practical for users to constantly carry around markers, nor is it practical to canvas the entire environment with hundreds of markers. Additionally, each marker-based AR application is designed around a specific marker or family of markers. A marker design distributed with a given application will generally not work with other unrelated AR systems.

Markerless AR attempts to solve these problems by doing away with markers and instead using visual features already present in the environment. Markerless AR systems can be broken down into two main categories. The first does not require markers in the traditional sense, but still depends on a pre-existing map of the visual features for the system to recognize [12]. The downside here is that should the environment ever be modified in any way, the map will no longer be consistent. The second type of markerless AR system attempts to map the environment online in real time, while at the same time rendering augmentations [14, 15]. These systems are capable of dynamically creating and updating a map, but typically rely on some sort of initial calibration step that requires user input.

The downside of markerless AR systems is the daunting computational requirements required by the various computer vision algorithms that comprise them. This issue is even more critical in handheld mobile devices, where processor speeds and memory are severely limited. We classify "handheld mobile devices" as smartphones, tablets, PDAs, etc. Laptop computers are not considered handheld in the context of this thesis, as they cannot be used with the same convenience as pocket-sized smartphones and tablets.

A further challenge when dealing with mobile hardware is the issue of power consumption, which is directly impacted by the computation workload taking place on the device. By increasing AR pipeline efficiency, we can in turn increase battery life while using the AR application.

Recent work has been able to achieve real-time performance on laptop computers using multi-threaded algorithms [15] or Graphics Processing Unit (GPU) acceleration [12]. While certain mobile devices are now equipped with onboard GPUs, they are not easily programmable and only the most recent generation of smartphones have multi-core processors. Therefore, a solution relying on brute force computing power is not suitable for these devices.

1.3 Objectives

The aim of this thesis is to prove that it is possible to perform robust, real-time, markerless AR on current generation mobile devices through the creation of a highly efficient AR pipeline tailored to mobile hardware. In the context of this thesis, real-time means that the user is not inhibited in any way by having to wait for offline processes to complete, and the system operates at a speed of at least 15Hz.

The proposed system is designed to be applicable to all AR application domains, including those identified in Section 1.1. The computational restrictions of mobile devices are addressed at each stage in the pipeline. We begin by analyzing and testing popular algorithms with the goal of finding which is best suited for mobile hardware, and then modifying or tuning them to maximize efficiency. The end result is a robust, mobile AR pipeline that is demonstrated on several mobile platforms at real-time performance of 15Hz.

1.3.1 Contributions

To the best of our knowledge, there currently exist no similar end-to-end AR systems for mobile devices that achieve this level of performance while incorporating all the functionality present in our system. In addition to an extensive survey of the algorithms relevant to the general AR pipeline, we make the following improvements and optimizations:

- A novel approach to the computation of a binary Bag-of-Words descriptor vocabulary through the use of k-medians clustering.
- Adaptive pose estimation based on whether the system is matching or tracking to balance execution speed and pose accuracy.
- Use of a constantly updating secondary ORB descriptor instead of computationally expensive image pyramids to achieve relative scale invariance. Our method adds a trivial amount additional computation.
- Hardware SIMD implementations of YUV to RGB image decoding and binary descriptor matching that yield an approximate twofold and threefold increase in efficiency respectively in comparison to non-SIMD implementations.

1.4 Thesis Organization

The remainder of this thesis is laid out in four additional chapters. Chapter 2 begins with a brief overview of computer vision then proceeds to describe the previous work done with respect to each of the components that make up the presented AR system. These components are condensed into three key areas: interest point detection and matching, image classification using Bag of Words (BoW), and pose estimation. Finally, previous work in both marker-based and markerless AR systems is discussed.

In chapter 3 the android development platform is discussed, as well as all other third party libraries that were used. An overview of the system and its implementation is provided, followed by an in-depth description of each component within the system.

Chapter 4 details the experiments and benchmarks used to test the system. Results are ranked quantitatively against previous work with respect to execution speed. Where applicable, other metrics such as accuracy are also be tested. Qualitative results are presented to illustrate what the user would actually see when using the system.

Finally, chapter 5 summarizes this thesis, the results that were obtained, and discusses some ideas for future research.

CHAPTER 2 BACKGROUND

In this chapter, previous work related to this thesis is discussed. It begins with a description of the background of the specific components that make up this and other markerless augmented reality systems, and concludes by describing similar complete marker-based and markerless AR systems incorporating some or all of these components.

2.1 Interest Point Detection and Matching

Interest point detection and matching forms the core of many popular computer vision algorithms and processes at the time of this writing. Interest points, in the general sense, are 2D image coordinates that are visually distinct in relation to their immediate surroundings. A good interest point remains visually distinct through transformations such as rotation, scale, shear, and changes in lighting.

2.1.1 Corner Detectors

One of the earliest and most well-known interest point detectors is the Harris corner detector [16]. Proposed by Harris and Stephens in 1988, the Harris detector measures a pixel's "corner-ness" by comparing a patch of pixels to nearby overlapping patches. A pixel at the centre of a patch that matches poorly to its surroundings is considered a good corner. This operation is made more efficient by analyzing the partial derivatives of pixel intensity values in the horizontal and vertical directions. This is essentially a measure of the rate-of-change in gradients. Pixels where strong horizontal and vertical gradients meet are considered good corner candidates. Figure 2.1 shows a sample image alongside its horizontal and vertical derivatives.

Once the derivative image has been computed, the corner-ness score of each pixel can be determined by analyzing the eigenvalues of the structure tensor matrix (Equation 1) created from an image patch centred on that pixel. Two small eigenvalues represent a poor interest point, one high and one low represents an edge, and two high eigenvalues represent a corner.

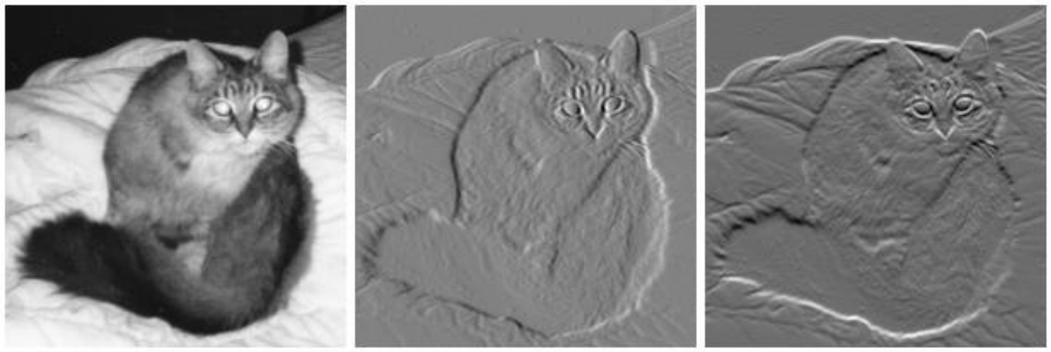

Figure 2.1 - A greyscale image and its horizontal and vertical derivatives (image found at [17]).

$$C = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}$$

Equation 1 - Structure tensor matrix.

To save on computation, the Harris detector approximates this eigenvalue response based on the determinant and trace. The Shi-Tomasi detector [18] computes this eigenvalue explicitly because it tends to produce more stable interest points. Other than this, the Shi-Tomasi detector is the same as the Harris detector.

Both the Harris and Shi-Tomasi corner detectors can be computationally expensive, even with eigenvalue approximations. The FAST (Features from Accelerated Segment Test) corner detector [19, 20] was introduced by Rosten and Drummond in 2005. To increase computation speed, it uses a much simpler method of finding corners than the

Harris or Shi-Tomasi detector. For each pixel in the image, the intensity values of the pixels in a bounding circle (usually a radius of three pixels) are compared. If there is a continuous portion of the bounding pixels (usually 9-12 of 16) that are either lighter or darker by a certain amount, that pixel is considered an interest point (Figure 2.2).

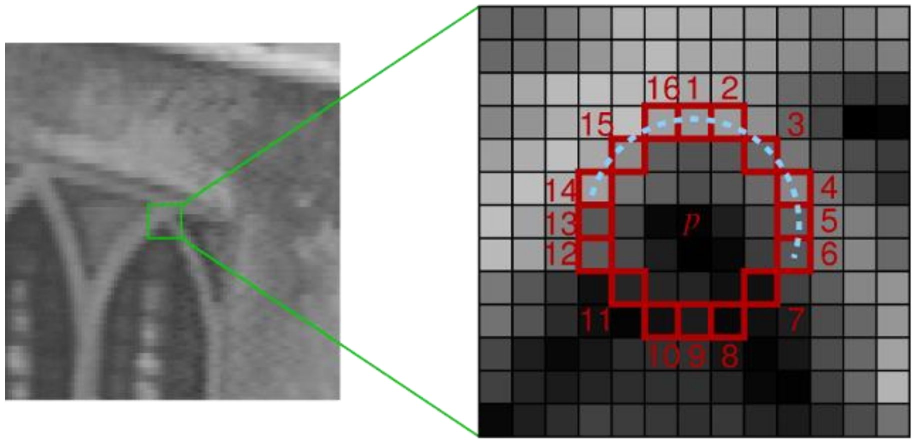

Figure 2.2 - Example of a FAST corner (image found in [20]).

2.1.1.1 Patch Correlation

The primary method of matching interest points before more robust alternatives were developed was through patch correlation. A square patch of pixels (7x7, 11x11, etc.) is extracted from around each interest point. A larger window provides more robust matching, but at the cost of increased computation time. Patches are matched to one another by comparing the greyscale intensities of two patches in various ways. Some of the common metrics for patch comparison include Sum of Absolute Differences (SAD), Sum of Squared Differences (SSD), and Normalized Cross Correlation (NCC).

Since each interest point is represented solely by a 2D image coordinate, patch correlation is only effective under simple image translations. This is useful for stereo camera configurations, but not when images are separated by rotation or scaling. With the exception of NCC due to the patches being normalized (Figure 2.3), these methods are not even robust to changes in lighting.

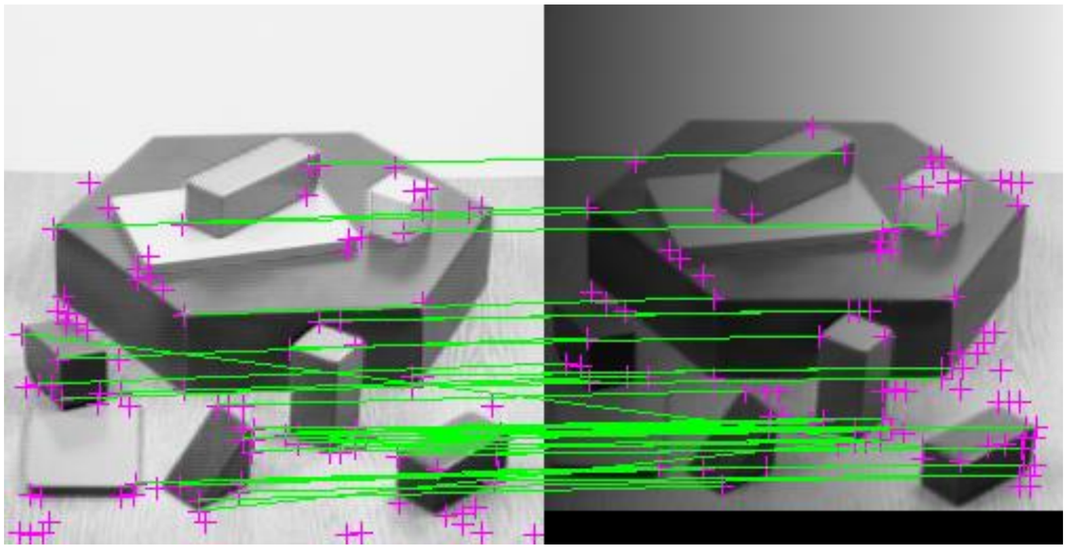

Figure 2.3 - Matching Harris corners using NCC under different lighting conditions (image courtesy of [21]).

2.1.1.2 Optic Flow

Optic (or optical) flow is a method of tracking interest points through camera motion rather than matching two interest points after that motion has taken place. More formally, we refer to *image flow* as the motion of the projection of true object motion onto the 2D image between image frames, or how an object is observed to move through the image itself. Optic flow is the process of observing the change in the image data with the goal of recovering the true image flow.

A method for globally calculating optical flow in the context of machine vision was presented by Horn and Schunck in 1981 [22]. The most popular combined interest point detector and tracker is the Kanade-Lucas-Tomasi (KLT) tracker, first presented by Tomasi and Kanade in [23] and based on the earlier work of Lucas and Kanade in [24]. In each new image frame, pixels are tracked within a search window based on the apparent velocity of their surrounding gradients. The KLT tracker begins by extracting Shi-Tomasi corners in an initial camera frame and then tracks those corners in subsequent frames (Figure 2.4). However, since each point is only tracked relative to its previous location, tracked points will eventually drift from their correct locations due to accumulated error.

Optic flow is most often used when visual input is in the form of a continuous video stream, or images with very small spatial separation. Larger spatial separation can be overcome by iteratively down-sampling the images until the separation is smaller, but this comes at the cost of increased computation. The great benefit of optic flow is that it can track interest points through almost any camera motion, as long as each intermediate transformation is small enough and the motion in the patch centred on each interest point is mostly uniform. Using optic flow it is possible to track points through image rotation, scaling, and even shear. This is a large improvement over patch correlation for non-sparse input data.

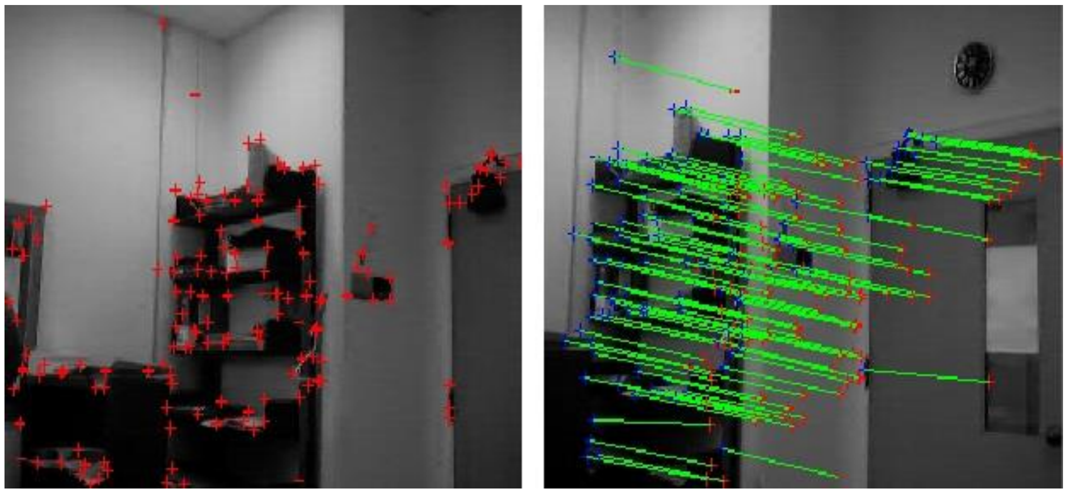
Figure 2.4 – Left: Harris corners. Right: Original corners in red, tracked corners in blue (image courtesy of [21]).

2.1.2 Feature Descriptors

In 1999, David Lowe published a seminal paper on a new method of feature detection and matching called Scale-Invariant Feature Transform (SIFT) [25]. SIFT introduced an invariant feature descriptor, a 128-dimensional floating point vector computed around a given interest point that is rotation, scale, and lighting invariant. To achieve these invariances, SIFT features are detected using a Difference-of-Gaussian method at multiple image scales and each feature is assigned an orientation based on its surrounding gradients. Descriptors are extracted by computing a histogram of gradients of an image patch centred on each feature. These patches are first aligned with the feature's orientation and scaled to a common size before the histogram is computed in order to ensure scale and rotation invariance. Figure 2.5 shows SIFT features drawn over an image, highlighting their size and orientation. By using descriptors that are unaffected by changes in scale and orientation, robust matching can be achieved over large camera transformations. The downside is that detecting SIFT features and extracting their descriptors is very computationally expensive.

The Speeded-Up Robust Features (SURF) [26] detector is another descriptor-based detector and attempts to improve on the high computation cost of the SIFT detector. The SURF descriptor is a 64-dimensional floating point vector, though some implementations optionally compute an expanded 128-dimensional descriptor vector. SURF features are based on the *Hessian matrix*, whose computation is sped up by using box filters to approximate the second order Gaussian derivative. Blob-like regions where the determinant of the Hessian matrix is a local maximum are considered potential interest points.

SURF also uses local image gradients to compute the descriptor, but relies heavily on integral images to speed up this process. At the time of publication, the authors of SURF reported an approximate 65% reduction[1] in computation time over SIFT while providing comparable scale, rotation, and lighting invariances. Despite this, SURF is still not suitable for real-time applications. Further speed-ups have been achieved by parallelizing the SURF algorithm through a GPU implementation [27], which finally achieves real-time performance on modern computers.

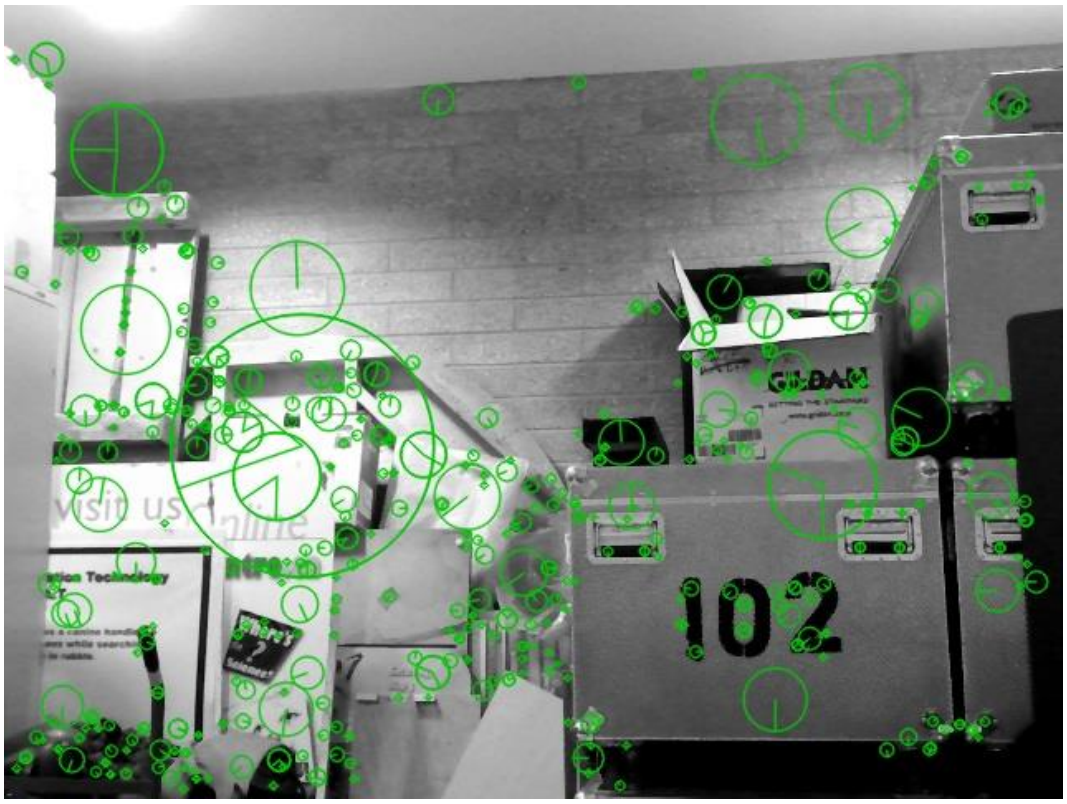

Figure 2.5 - Example of SIFT features with size and orientation data.

[1] At the time of this thesis writing, there are numerous publicly available implementations of both SIFT and SURF. Execution speed depends largely on which implementation is being used.

Both SIFT and SURF use floating point descriptor vectors, which are costly to compute despite GPU acceleration. Floating point arithmetic is more computationally expensive than integer or binary operations, especially on mobile or embedded hardware. In order to overcome this, Binary Robust Independent Elementary Features (BRIEF) were introduced by Calonder et al in 2010 [28]. Each BRIEF descriptor is a 32-byte binary string, making it significantly smaller than the SURF or SIFT descriptors which require 256 and 512 bytes respectively. The BRIEF algorithm avoids floating point arithmetic entirely by using simple comparisons of pixel intensity values when computing the descriptor. The BRIEF algorithm does not include interest point detection, but requires existing interest points such as FAST or Harris corners to be provided. Although BRIEF descriptors are computed approximately 25x faster than SURF descriptors, they lack both rotation and scale invariance.

Finally, the Oriented FAST and Rotated BRIEF (ORB) descriptor was introduced in 2011 by Rublee et al [29]. As the name suggests, the ORB algorithm adds an orientation component to extracted FAST corners, and computes a rotationally invariant version of the BRIEF descriptor. The ORB descriptor can be computed as fast as the BRIEF descriptor, with the rotational invariance of SIFT and SURF. The one notable weakness to the ORB descriptor when compared to SIFT and SURF is that it still lacks scale invariance. Recent implementations of ORB have added scale invariance by finding features at different image scales similar to SIFT and SURF, but this slows down detection speed dramatically.

Attempts have been made to adapt SIFT and SURF to mobile devices. Wagner et al proposed in [30] a hybrid between FAST corners and a reduced version of the SIFT descriptor. Rather than compute descriptors every frame, they track existing features using SAD patch correlation. This method achieves upwards of 20Hz while extracting ~150 features per frame (320x240). Chen et al propose a modified version of SURF in [31] that achieves a roughly 30% speed-up over the original SURF algorithm, but nevertheless falls far short of real-time operation by requiring over 400ms to extract 300 features on their test device.

2.1.2.1 Descriptor Matching

Modern approaches to interest point matching rely on descriptor comparison over image patch correlation. Descriptors are matched by comparing the distance between them using some metric. In the case of floating point descriptors (SIFT, SURF), the Euclidean distance is used (Equation 2). The smaller the distance between two descriptors, the more similar they are. Given two sets of descriptors, the best match in the first set is the descriptor in the second set that yields the lowest distance. Exhaustively matching every descriptor in the first set to every descriptor in the second set will return the most matches, but this “brute force” matching is a very costly operation. Approximate Nearest Neighbour (ANN) methods like the one found in [32] are often used to reduce the search space for each descriptor by pre-sorting them into tree structures, which can greatly improve matching speed but comes at the cost of producing fewer matches as they are not guaranteed to find the best match for every descriptor. A GPU-accelerated brute force matching method was proposed in [12] for matching SURF descriptors. This method places the two sets of descriptors in a row and column matrix respectively and performs a large matrix multiplication to compute the dot product for every pair of descriptors. Since SURF descriptors are normalized, descriptor pairs yielding a dot product nearest to 1.0 are the best match. This method achieves very robust matching while maintaining high execution speed by using GPU acceleration.

$$Distance = \sqrt{\sum_{n=1}^{64}(Desc1_n - Desc2_n)^2}$$

Equation 2 - Calculating Euclidean distance between two SURF descriptors.

In the binary domain (BRIEF, ORB), the Euclidean distance cannot be directly applied to binary strings so the *Hamming Distance* is used instead to measure the difference between two descriptors. The Hamming distance is defined as the number of bits that differ between the strings. This can be computed very quickly by performing an exclusive-or (XOR) operation on the two strings, and then counting the number of bits that are set (Equation 3). Logical operations such as XOR are very fast, and most processor architectures have a built-in instruction for counting the number of 1s in a binary string (known as a *population count* instruction). This makes matching binary descriptors very fast in comparison to matching floating point descriptors.

$$Distance = POPCNT(Desc1 \wedge Desc2)$$

Equation 3 - Calculating Hamming distance between two binary strings.

Similar tree-based matching methods for binary descriptors have been proposed, most notably the hierarchical clustering tree method in [33]. This method clusters binary descriptors around randomly selected centres in order to reduce the search space of the descriptor set. Multiple tree layers with their own set of centres can be used, each layer further reducing the search space. Descriptors are matched through the nodes of the tree, always taking the path whose centre yields the best match until a leaf is reached. This leaf is a small subset of the original descriptor set, and it is to this set that the incoming descriptor is matched.

2.2 Image Recognition using Bag of Words

The matching methods discussed so far apply to matching features between two images. However, in certain applications (image recognition and classification), there is a need to compare one image to multiple images. It is not feasible to compare millions of descriptors, and even tree-based matching methods become too expensive and cannot be performed in real-time.

The Bag of Words (BoW) model was originally developed to classify text documents based on a small vocabulary of key words [34]. This model has since been adapted to visual features [35, 36], where the vocabulary of words is replaced by a vocabulary of image descriptors. In order to build a vocabulary of descriptors, k-means clustering is often used to segment a set of SIFT or SURF descriptors. The centroid descriptors of these segments represent the descriptor vocabulary. Depending on the distribution of best matches to this vocabulary, it is possible to determine which image the new frame best matches to. Once this has been determined, the two frames are matched to one another as usual.

K-means clustering is not directly applicable to binary descriptors because there is no clear definition of the mean of two binary strings. However, the bag of words model was applied to binary BRIEF descriptors by Galvez-Lopez and Tardos in [37]. They report using traditional k-means to cluster the descriptors and then rounding the resulting centroids to move back into the binary domain.

2.3 Pose Estimation

Given a series of 2D image points and their corresponding 3D world points, it is possible to determine the location and orientation of the camera (pose) in world coordinates. This is known as the *Perspective n-Point* (PnP) problem. The camera pose can be used along with the camera intrinsic parameters (focal length, distortion parameters) to create a projection matrix (Equation 4). An accurate projection matrix is important in augmented reality applications because it is used to project 3D objects into the 2D camera scene. An inaccurate pose can cause excessive augmentation jitter, leading to a poor user experience. There are many different algorithms for solving the PnP problem including the iterative methods POSIT [38] and CamPoseCalib (CPC) [39], both of which are evaluated in [40]. Another popular, more recent method is known as Efficient Perspective n-Point (EPnP) [41], which is a non-iterative algorithm that solves the PnP problem in linear time. Both the EPnP and iterative methods can handle planar and non-planar correspondences.

$$P = \begin{bmatrix} F_x & 0 & C_X \\ 0 & F_y & C_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{0,0} & R_{0,1} & R_{0,2} & t_{0,0} \\ R_{1,0} & R_{1,1} & R_{1,2} & t_{1,0} \\ R_{2,0} & R_{2,1} & R_{2,2} & t_{2,0} \end{bmatrix}$$

Equation 4 - Building projection matrix from camera pose.

2.4 Marker-Based AR

In 2003 the AR-Toolkit project was ported to the Pocket PC by Wagner et al [42], and the first self-contained AR system to run on a consumer-grade cell phone was presented in 2004 by Mohring et al [43]. Their system recognizes different markers via circular bar codes and detecting gradient changes in the red, green, and blue colour channels. Their entire pipeline achieved a frame rate of 4-5fps, at a camera resolution of 160x120.

The progression of marker-based AR is not strictly relevant to this thesis, but at the time of this writing there are numerous publically available marker-based AR applications and games for mobile platforms.

2.5 Markerless Augmented Reality

The best known work in monocular camera localization and mapping is the MonoSLAM system developed by Davison et al [44]. The authors successfully applied Simultaneous Localization and Mapping (SLAM) methodologies to the vision domain using a monocular camera as the sole input device. MonoSLAM maintains a sparse map of oriented planar textures (~12 for any given camera view) centred on Shi-Tomasi corners. The system is initialized using a planar target containing four known features before dynamically updating the map. They present results in the AR domain and achieve real-time operation of 19ms per frame on a 1.6 GHz computer while rendering augmentations.

Also well-known is the Parallel Tracking and Mapping (PTAM) system, introduced by Klein and Murray in 2007 [15]. PTAM uses a multi-threaded approach to simultaneously track interest points by morphing and matching image patches, and maintains a map of these patches. PTAM uses a stereo pair of images of a planar scene for initialization, and uses bundle adjustment to determine and update the 3D locations of interest points. They achieve tracking performance of approximately 20ms per frame on a 2.66 GHz dual-core computer.

Taehee and Hollerer also propose a multi-threaded approach in [45]. They use a hybrid tracking method which extracts SIFT features (instead of image patches) and then tracks them using optic flow. They achieved real-time performance by only extracting and matching SIFT features periodically in a thread separate from the tracker. They also perform scene recognition by matching to previously recorded SIFT features. No mention is made of bag-of-words or any other recognition technique. Their results are obtained on a laptop computer, but due to the usage of SIFT it is unlikely that their system would translate well onto mobile phones and tablets.

The feature extraction and tracking technique used by Taehee and Hollerer above is a common methodology in markerless AR systems. Rather than performing computationally expensive feature detection in every frame, features are extracted from a single frame and good matches are tracked through as many frames as possible before detection must take place again. This is commonly referred to as having a *tracking* mode and a *lost* or *searching* mode. When feature matching fails and too few correspondences are found, the system is considered lost and must continue extracting and matching features. When a sufficient number of good matches are found, the map scene is present in the camera frame and the system enters tracking mode. Tracking continues until too few tracks remain to obtain a reliable pose, or the map scene leaves the camera view. However, recent advances in hardware have allowed some markerless systems to break away from this paradigm.

GPU acceleration is a common technique to speed up feature detection and matching. In [12], Tam and Fiala remove the tracking step entirely by using the GPU implementation of SURF from [27] and brute force descriptor matching through a large matrix multiplication also performed on the GPU. In this way, the system is fast enough to perform feature detection and matching on every frame while maintaining an operation speed of 10-12Hz. This technique is not currently applicable to mobile devices, however, as they tend not to have powerful, programmable GPUs. Therefore, most mobile markerless AR systems still make use of the hybrid lost/tracking framework.

2.5.1 Markerless AR on Mobile Devices

A reduced version of the PTAM system has been adapted to the iPhone [14], but results showed severely reduced accuracy and execution speed. PTAM is intended for use in small AR workspaces, and suffers reduced performance as the map gets bigger and bigger due to the bundle adjustment process being cubic with respect to the number of features in the map ($O(n^3)$).

As a continuation of their work in [30], Wagner et al propose a more complete version of their feature detection and matching system in [46]. They use a similarly modified version of SIFT and outlier rejection that runs at approximately 26Hz. However, when AR-related overhead (image retrieval from camera, rendering, etc.) is taken into consideration, the speed drops to 15Hz. They combine this with patch tracking to greatly improve speed to 8ms per frame (not including AR overhead) instead of using SIFT on every frame. Their system runs on 320x240 imagery during the SIFT phase, and down-samples further to 160x120 while tracking. In addition, a maximum of 100 features are tracked at any given time.

Another way to achieve high performance on mobile devices is to outsource computation to a more powerful server wirelessly. The AR-PDA project presented in [9] sends live camera imagery to a server which recognizes 3D objects in each image, overlaying augmentations before sending it back to the Personal Digital Assistant (PDA). Another system that offloads computation was developed by Hile and Borriello as a navigation aid [47]. Their system involves sending live imagery of a user's surroundings from a personal device to a server, which matches the imagery to a known floor plan. The server then overlays directions for the user to follow and returns the augmented image (Figure 2.6). The server is able to reduce the floor plan search area using knowledge of which Wi-Fi node the user is nearest to. This of course requires a significant amount of pre-existing infrastructure in addition to a computation server. Their goal was five seconds or less to receive directions from the server, and their reported results range from 5-30 seconds. Because of the unpredictability of publicly available Wi-Fi, this type of approach is only suitable for very specific application domains.

This concludes the description of previous work. To the best of our knowledge, there exists no complete AR system for mobile devices that incorporates all the elements presented in this thesis without requiring a significant amount of pre-processing. Most notably absent from existing systems are online map creation and multiple map support. The next chapter discusses the technical details and implementation of our AR pipeline.

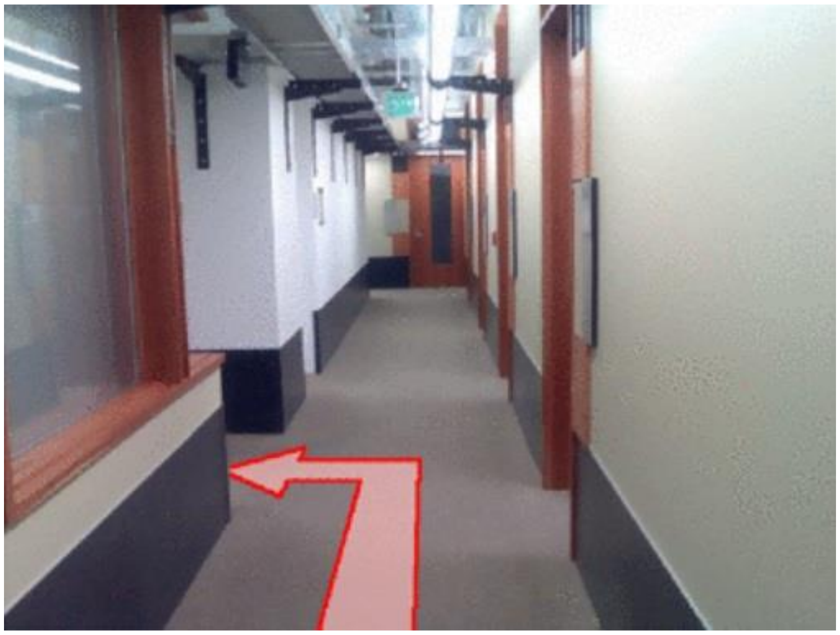

Figure 2.6 - From [47], navigation information overlaid by a computation server.

CHAPTER 3 TECHNICAL APPROACH

3.1 Introduction

In this chapter, the implementation of the system created for this thesis is discussed. Overviews of the map creation process and the AR pipeline are presented in section 3.3. The Android platform and other elements common to both map creation and the AR pipeline are discussed in section 3.4, followed by in-depth descriptions of these processes in sections 3.5 and 3.6. Modifications to standard algorithms are described, as well as how each algorithm is tuned for best performance within the system. First, however, we begin with a brief description of how a user would experience the system.

3.2 User Description

From the point of view of the user, the system begins by pointing the camera at a textured surface from which a map can be created. Additional maps can be created at the request of the user throughout the operation of the system. Once at least one map has been created, the user can point the camera at the textured surface used to create the map and the system will recognize and match to it. If the matching is successful, augmentations will be rendered over the textured surface. The user is able to point the camera at any of the map textures that have been added, and unique augmentations will be drawn depending on which map surface is being viewed.

3.3 Map Creation and AR Pipeline Overview

Before the AR pipeline can begin, at least one map must be created. A map is defined simply as a set of descriptors with corresponding 3D world coordinates to which live, incoming camera frame descriptors are matched. The workflow of this process is shown in Figure 3.1. The system begins by obtaining a camera image from the live video feed and *decoding* it from the YUV colour space as provided by the camera into both the greyscale and RGB colour spaces. The RGB image is displayed on the screen, and the user is asked to select a Region of Interest (RoI) from this image. ORB features are extracted from this region, and are projected into 3D space using the camera's intrinsic parameters[2] in order to establish the world coordinate system. The map descriptors are clustered into a tree, and a representative descriptor vocabulary is created to facilitate efficient map lookup. These map elements are used as input for the AR pipeline described below.

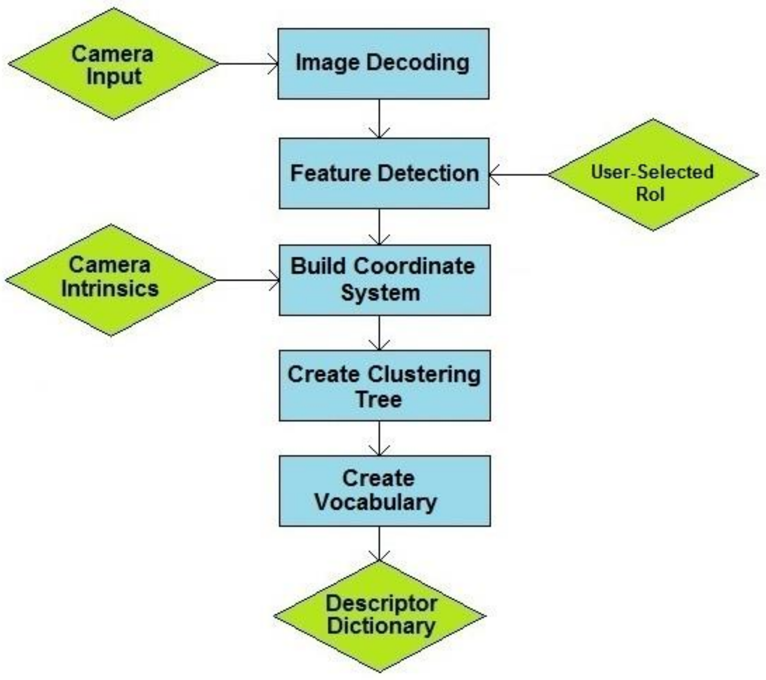

Figure 3.1 – Map creation overview. A map image can be provided to the system or created online from live camera imagery at the request of the user.

[2] Camera intrinsic parameters, or "intrinsics", include horizontal and vertical focal lengths and distortion coefficients that are computed in an offline calibration process.

An overview of the AR pipeline is shown in Figure 3.2. Live camera images are decoded in the same manner as map creation. The greyscale image is used in the remainder of the pipeline, and the RGB image is displayed on-screen. If features from the previous frame are not being tracked, the system extracts new features and attempts to match them to a map by looking them up in the descriptor dictionary (output of the map creation process). If a map is found, the new features are matched to that map, and the camera's pose is calculated from those matches. Using the pose, augmentations are drawn. If the system is in tracking mode, existing feature tracks are used to compute the pose directly instead of extracting and matching new features.

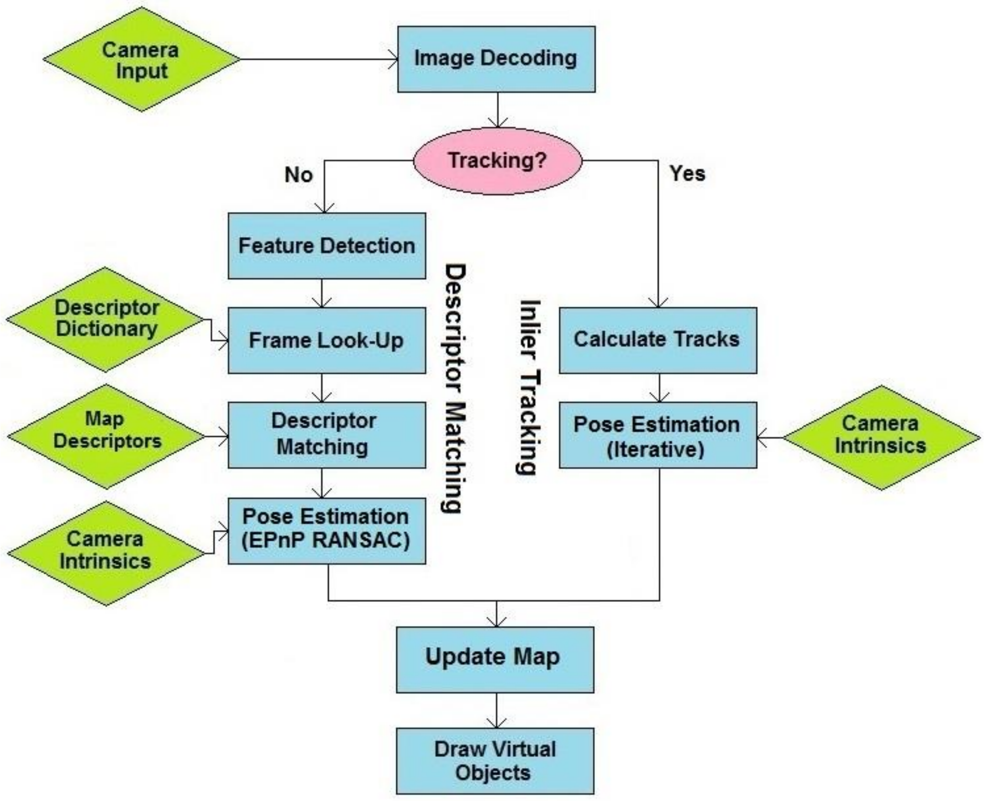

Figure 3.2 - AR pipeline overview.

Image decoding, feature detection, and descriptor matching are used in both the map creation process and the main AR pipeline. Therefore, these algorithms are described first in the general sense beginning in section 3.4.4. A more complete description of how these algorithms are used is found in the map creation and AR pipeline sections (3.5 and 3.6 respectively).

3.4 Android Development Platform

Android is an open source, Linux-based operating system for mobile devices. It was chosen as the platform for this system because of its availability on a wide range of devices of varying processor speeds and hardware capabilities (as opposed to iOS, which is only available on Apple devices). Android applications are written primarily in Java, but it is also possible to integrate C/C++ code. For this implementation, all the display and user interface elements are written in Java, while the heavier visual processing elements are written in C/C++. This was done to take advantage of pointer arithmetic and obtain direct access to image pixel information without having to use access functions.

3.4.1 Hardware Limitations

Many mobile devices do not have a hardware Floating Point Unit (FPU), making floating point arithmetic very inefficient. ARMv7 series processors include hardware FPUs, whereas v5 and v6 processors do not. Any device without a hardware FPU will see a disproportionate slowdown in performance when executing code with a large amount of floating point calculations. Therefore it is important to use integer arithmetic whenever possible when dealing with mobile hardware - a fact taken into consideration when developing this system.

A common method to compensate for limited hardware capability is to down-sample imagery before it is processed (sometimes as low as 160x120). The problem with down-sampling, however, is that in a complex scene, large amounts of detail can be lost. All imagery used by this system is 640x480, which is a very common resolution used in computer vision. This resolution is supported natively by most cameras, meaning no software image resizing takes place.

3.4.2 NEON Instruction Set

Devices using the ARM Cortex-A series of processors can make use of a set of assembly language primitives called the NEON instruction set. These primitives use ARM's *Single Instruction, Multiple Data* (SIMD) engine, providing a powerful way to speed up parallelizable algorithms. The SIMD engine contains 32 registers, each of which is 64-bits wide. Alternatively they can be used as 16, 128-bit wide registers.

Since many computer vision algorithms perform operations on each pixel of an input image independently, we make a contribution to the efficiency of the pipeline by using NEON instructions to process multiple pixels or bits simultaneously. In particular, NEON functions are implemented for image decoding and binary descriptor comparison[3].

3.4.3 Libraries Used

This system makes use of Willow Garage's OpenCV library for Android [49]. It provides several useful data structures as well as implementations of the ORB descriptor extractor, the iterative and EPnP pose estimation algorithms, and an algorithm for calculating optic flow. The system also makes use of the OpenGL graphics library in

[3] This precludes non-ARM based or pre-NEON ARM devices from benefiting from these speedups, but as ARM currently holds 95% of the smartphone market [48], we feel this is a fair restriction.

order to render and display the augmentations. All other components and algorithms present in this system were implemented from scratch for the purpose of this thesis.

3.4.4 Image Decoding

The conversion routine for YUV to RGB was implemented using the NEON assembly primitives. When the raw image stream first comes from the camera, it is in the YUV420sp (YUV420 semi-planar) colour format. In this format, each pixel has a luminance value and two chrominance values; however, each set of two chrominance values belong to a block of four pixels (Figure 3.3).

Y_1	Y_2	Y_3	Y_4	Y_5	Y_6
Y_7	Y_8	Y_9	Y_{10}	Y_{11}	Y_{12}
Y_{13}	Y_{14}	Y_{15}	Y_{16}	Y_{17}	Y_{18}
Y_{19}	Y_{20}	Y_{21}	Y_{22}	Y_{23}	Y_{24}
U_1	V_1	U_2	V_2	U_3	V_3
U_4	V_4	U_5	V_5	U_6	V_6

Figure 3.3 - YUV420sp data format. Colour coding links luminance values with their chrominance counterparts.

The luminance value of each pixel can be also be used as a greyscale intensity, making it very easy to create a greyscale image from the YUV data. This is very convenient because the vast majority of computer vision algorithms operate on greyscale images only, including those present in the remaining sections of the system. However, in order to draw the camera image as a background texture in OpenGL, the YUV420sp image must also be converted into an RGB image (Equation 5) [50].

$$\begin{aligned}
\boldsymbol{R} &= \mathbf{1.164}(\boldsymbol{Y} - \mathbf{16}) + \mathbf{1.596}(\boldsymbol{V} - \mathbf{128}) \\
\boldsymbol{G} &= \mathbf{1.164}(\boldsymbol{Y} - \mathbf{16}) - \mathbf{0.813}(\boldsymbol{V} - \mathbf{128}) - \mathbf{0.391}(\boldsymbol{U} - \mathbf{128}) \\
\boldsymbol{B} &= \mathbf{1.164}(\boldsymbol{Y} - \mathbf{16}) + \mathbf{2.018}(\boldsymbol{U} - \mathbf{128})
\end{aligned}$$

Equation 5 - Conversion formulae for YUV to RGB.

Optimizations were made to the above formula to avoid performing floating point arithmetic. A factor of 64 was applied to each coefficient and then the coefficient was rounded to the nearest integer (Equation 6). After performing the conversion, this factor is divided out. By using a power of two as the factor, the division can be done very quickly as a bit shifting operation. Greater conversion accuracy could be achieved by using a larger factor, but the value 64 was chosen because it allows every multiplication result to be represented as a short integer (2-byte) whereas the next highest power of two does not. This is significant because it allows twice as many pixels to be processed simultaneously through the NEON SIMD engine (eight 16-bit short integers as opposed to four 32-bit integers in a 128-bit double register).

$$\begin{aligned}
\boldsymbol{R} &= [\mathbf{75}(\boldsymbol{Y} - \mathbf{16}) + \mathbf{102}(\boldsymbol{V} - \mathbf{128})] / \mathbf{64} \\
\boldsymbol{G} &= [\mathbf{75}\,(\boldsymbol{Y} - \mathbf{16}) - \mathbf{52}(\boldsymbol{V} - \mathbf{128}) - \mathbf{25}(\boldsymbol{U} - \mathbf{128})] / \mathbf{64} \\
\boldsymbol{B} &= [\mathbf{75}\,(\boldsymbol{Y} - \mathbf{16}) + \mathbf{129}(\boldsymbol{U} - \mathbf{128})] / \mathbf{64}
\end{aligned}$$

Equation 6 - YUV to RGB conversion formulae using a reduced integer approximation.

The converted RGB values are packed into the RGB565 format. This format was chosen because it is compatible with the OpenGL graphics library on mobile devices and makes the texture half the size of the traditional ARGB8888 format. In ARGB8888, each value of red, green, and blue take eight bits, plus another eight bits for the alpha channel. In RGB565, the red value is 5 bits, green is 6 bits, and blue is 5 bits with no alpha channel. Since each new camera image needs to be uploaded through OpenGL, it is important to keep the texture as compact as possible.

Even though the resolution of the conversion formula was reduced and the image is stored with reduced colour depth, the image displayed on the screen is visually indistinguishable from an image converted using the traditional floating point equations and full colour depth.

3.4.5 Feature Detection

The system uses the OpenCV implementation of the ORB feature detector. First, FAST corners are found and then ranked using the FAST or Harris score. Although the FAST score is faster to compute, the Harris score was chosen for this system because it tended to produce more stable interest points. That is, interest points were more consistently detected at good corners when filtered using the Harris score. When using the FAST score, it was common to see interest points detected along edges as well as corners. Once interest points are found, ORB descriptors are computed for each point.

The ORB descriptor was chosen for two key reasons. First, it is invariant to rotation and lighting changes, and fairly robust to shear as well. This allows for matching under a wide range of viewing angles and orientations. The second reason is speed. Although real-time performance has been achieved with GPU implementations of SURF as in [12], mobile devices by and large do not have programmable GPUs on-board. Furthermore, each ORB descriptor is a binary string rather than a floating point vector as found in SIFT and SURF. This makes the descriptors very fast to compute and match.

3.4.6 Descriptor Matching

As mentioned previously, the problem with calculating the Euclidean distance is that it requires a significant number of floating point operations, including square roots. On the other hand, the Hamming distance between binary descriptors is very fast to calculate on all platforms. Furthermore, The NEON instruction set contains primitives for XOR and population count, and was used to further speed up descriptor comparison.

Matching between two groups of descriptors, A and B, involves finding for each descriptor in A the descriptor in B that yields the lowest Hamming distance. An absolute threshold of 70 is used on the Hamming distance to filter out poor matches. That is, no more than 70 of the 256 bits in the descriptor may be different. This threshold is based on the matching results presented in [51], and then confirmed empirically as a good value to filter out the most false positives without rejecting too many good matches. An additional criterion of uniqueness for a match uses the ratio between the first and second best matches. If the Hamming score of the best match is less than 70% of the second best match, it is saved (Equation 7). Otherwise the match is considered ambiguous and is rejected.

$$\frac{Score_1}{Score_2} < 0.7$$

Equation 7 - Acceptance ratio for the two best matches.

The final step in match filtering is to remove redundant matches. It is possible for a single map descriptor to be the best match for multiple frame descriptors. Because a single point in one image cannot correspond to multiple points in another image, only one of these matches can possibly be correct. The match with the best (lowest) Hamming score is accepted while the others are rejected.

Figure 3.4 shows three examples of ORB features being matched under the presented matching constraints. For clarity, at most 100 features are matched and drawn. In reality, 300-500 features are extracted from each camera frame. Even though ORB lacks scale invariance, our matching constraints are able to filter out all the false matches between the full-sized image and the scaled image, leaving a solitary correct match.

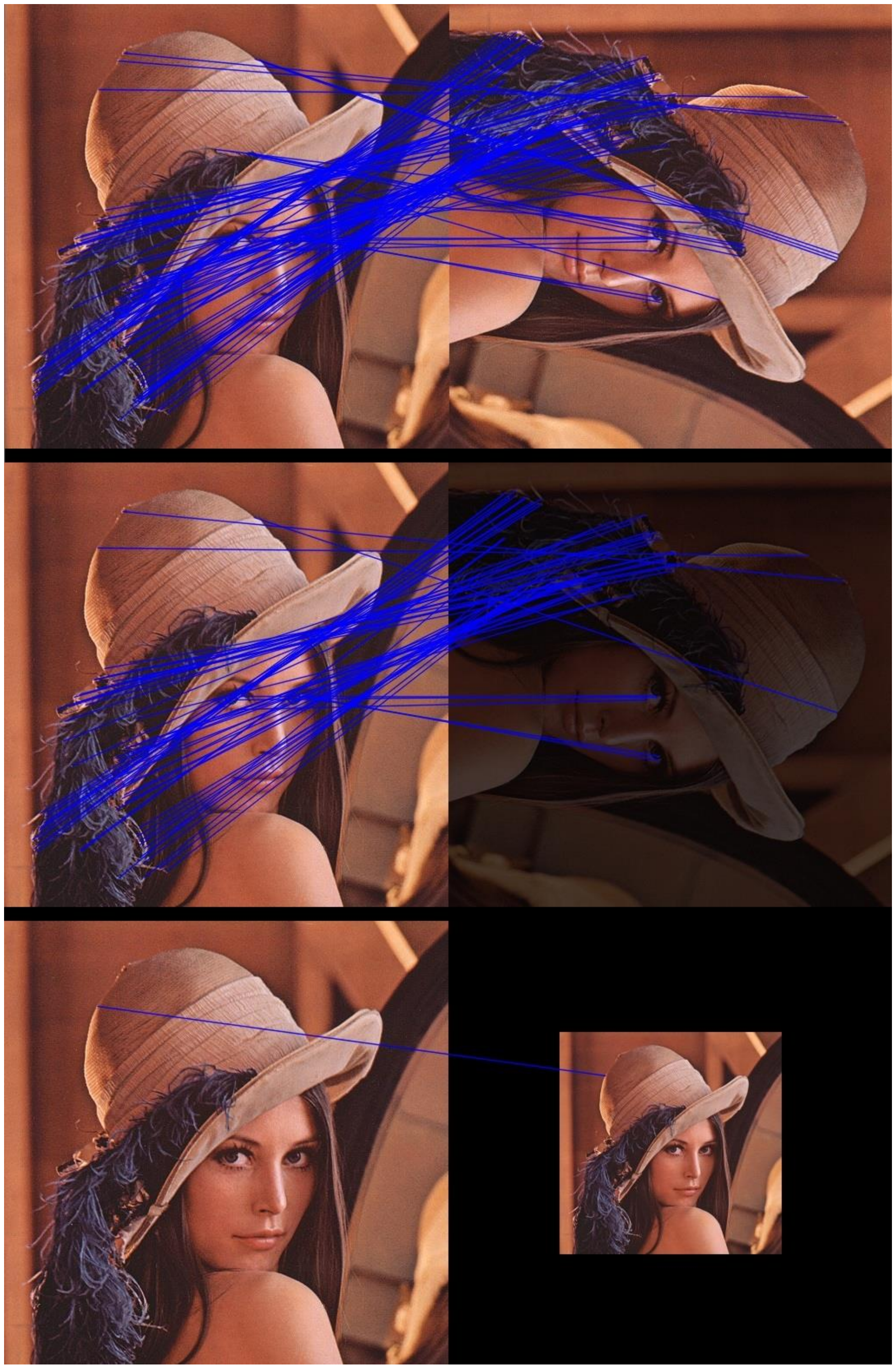

Figure 3.4 – Matching ORB descriptors. Top: 90-degree rotation. Middle: rotation and light reduction. Bottom: 50% scaling (image from [52]).

3.5 Map Creation

Map creation is the process of creating a set of reference descriptors that the system attempts to recognize later, and the map itself is the set of these reference descriptors and their 3D world coordinates organized into a tree structure. Map creation begins when the system is supplied with a reference image, either provided at start-up or by the user selecting a RoI from a live camera feed. In the latter case, the camera feed is displayed on the screen of the android device via an OpenGL texture. Once the user has defined the RoI, the corners of this region are converted from screen coordinates into camera image coordinates (Equation 8) and 3D OpenGL coordinates (Equation 9).

$$(u,v)_{image} = (u,v)_{screen} * \frac{(C_x, C_y)_{image}}{(C_x, C_y)_{screen}}$$

Equation 8 - Equation to convert screen coordinates to camera image coordinates. (Cx,Cy) refers to the centre of the image or screen.

$$z_{3D} = arbitrary$$

$$(x,y)_{3D} = \frac{\left[(u,v)_{image} - (C_x, C_y)_{image}\right] * z_{3D}}{(F_x, F_y)_{image}}$$

Equation 9 - Equation to project camera image coordinates into 3D OpenGL space. (Fx,Fy) refers to the focal lengths of the camera.

Map features are assumed to lie on a plane that is parallel to the image plane at the time of map image acquisition; a limitation imposed to avoid the user having to perform a stereo initialization step (as is the case with the PTAM system) and therefore keep map creation as user-friendly as possible. Since each map exists in its own independent coordinate system, the value of Z_{3D} in Equation 9 can be anything. It does not correspond to any real world measurements, but rather represents the number of arbitrary units from the camera centre to the map region in the camera image. This value is then used to calculate the X and Y coordinates and therefore determines the scale of the augmentation coordinate space (world coordinates). ORB features are then extracted from the RoI (500 when creating a map) and 3D coordinates are computed for each feature in the same way. The final product is a list of ORB descriptors that each correspond to a 3D world coordinate. At the time the map is created, each 3D coordinate corresponds to a single descriptor. However, during the map update step described in section 3.6.3, additional descriptors may be added to existing 3D points.

3.5.1 Map Challenges

At this point the map is simply a list of 500 ORB descriptors and their associated 3D points. This information alone is sufficient for matching to incoming camera frames and obtaining pose. However, to differentiate between multiple maps a more complex approach is required. Before the descriptors of an incoming camera frame can be matched to the descriptors of a given map, it must first be determined *which* map is actually present in the frame (Figure 3.5). This is done using the bag of words approach described in section 3.5.3. Once a candidate map has been selected using bag of words, the camera frame descriptors are matched to it. To avoid the use of brute force matching we employ a tree-based method described in section 3.5.2 below.

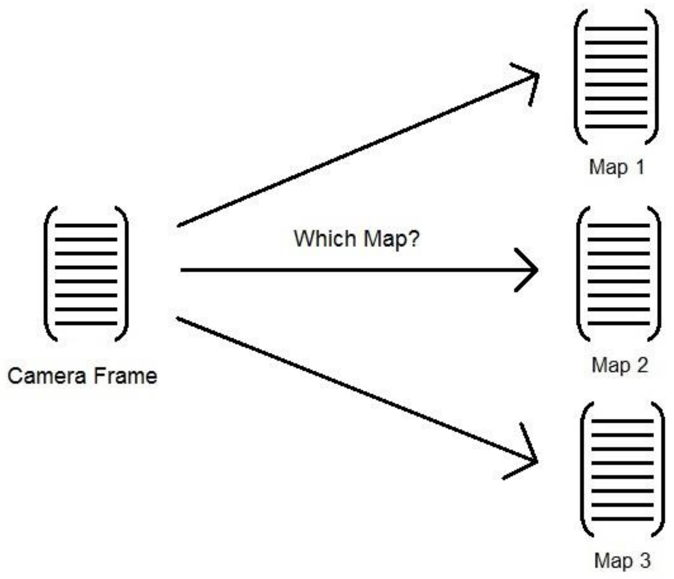

Figure 3.5 – Deciding which map to match the incoming camera frame to.

3.5.2 Descriptor Clustering Tree

Descriptor matching needs to be performed for every camera frame until a map is found and tracking begins. This must be done in real-time, making brute force matching impractical. Based on the work in [33], a clustering tree is used to speed up descriptor matching between the map and incoming camera frames. In the cited work, multiple tree layers are used to organize millions of descriptors. In the case of this system, it was found that a single tree layer with eight branches was sufficient due to the maps only containing 500 descriptors each.

To create this tree, eight descriptors are picked at random from the descriptor set to act as cluster centres. Each map descriptor is placed in one of these clusters, based on whichever centre it best matches to. This creates a set of eight descriptor sub-lists, each with an associated centre. No match filtering takes place here, the cluster chosen is simply the best match regardless of how high the Hamming score is. Figure 3.6 illustrates this process.

To match an incoming camera frame to this tree, frame descriptors are compared to each of the eight random centres. Brute force matching then takes place between each descriptor and the cluster whose centre yielded the lowest Hamming score. By using this method, frame descriptors need only be compared to a small subset of map descriptors, thus greatly improving matching speed.

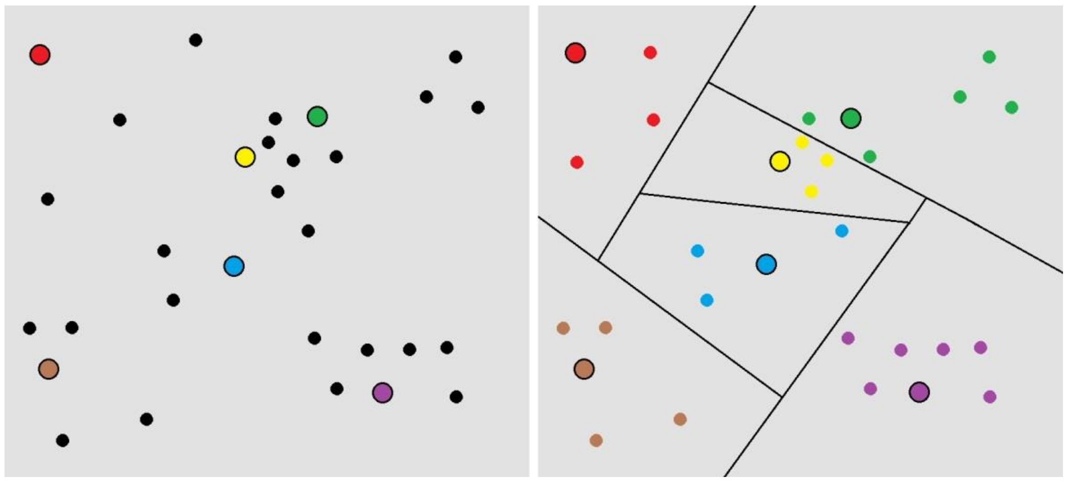

Figure 3.6 - Descriptor clustering about randomly picked centres. Left: random centres are picked from the descriptor set (colours). Right: descriptor set is clustered around centres. Each descriptor is assigned to the set defined by the closest centre.

It must be noted that while a linear brute-force matching of every frame descriptor to every map descriptor is guaranteed to find the best match, using a clustering tree is not. There is a chance that the best match for a particular descriptor is not found in the cluster whose centre it best matched to. In these cases there is the potential for a false match, but the speed-up achieved by using the clustering tree far outweighs the increase in the number of false matches.

3.5.3 Bag of Features Map Representation

When multiple maps are being used, the system must first decide which map if any is present in the camera frame before attempting to match descriptors. The simplest way is to simply match to every map and pick the one that matches best to the current camera frame. However, this very quickly becomes computationally impractical as the number of maps increases. Simply put, matching to *N* maps instead of a single map results in an *N*-fold increase in computation. Instead, a bag of features approach is used. Here, we make a contribution through a novel method for computing a binary descriptor vocabulary. Our method is described below.

First, a set of 100 random ORB descriptors is generated. The combined sets of descriptors from all maps are clustered around these random descriptors, similar to the clustering tree described previously. However, unlike the randomly picked static cluster centres used in the clustering tree, a process called *k-medians* [53] iteratively adjusts the cluster centres to better represent the distribution of the map descriptors. Note that clustering is an NP-hard problem, and we are not guaranteed to find a globally optimal solution.

The difference between k-medians and traditional k-means is the method used to compute the centroid of a group of binary descriptors. The value of each bit in the centroid descriptor is computed independently as the median of that particular bit in every other descriptor in the cluster. Once the new centroids have been calculated, the descriptors are then re-clustered around them. This process is repeated until subsequent centroids differ by an amount smaller than some threshold (Hamming distance) or a maximum number of iterations are performed. It was found that after 10 iterations, the

Hamming difference between subsequent centroids was nearly always below 10. Therefore the maximum number of iterations and the exit threshold were both set to 10. The centroid descriptors resulting from k-medians clustering form what is called the *vocabulary*, and each descriptor in the vocabulary is called a *word*.

Each map is matched to the newly created vocabulary. Map descriptors are placed into bins corresponding to which word they best matched to, creating a 100-bin response histogram (Figure 3.7). Each map's response histogram is normalized to prevent the number of map descriptors from impacting the shape of the histogram. This becomes important later when comparing histograms, since not every map and incoming camera frame is guaranteed to have the same number of descriptors. Each time a new map is added (based on user input), the vocabulary and map histograms are re-computed.

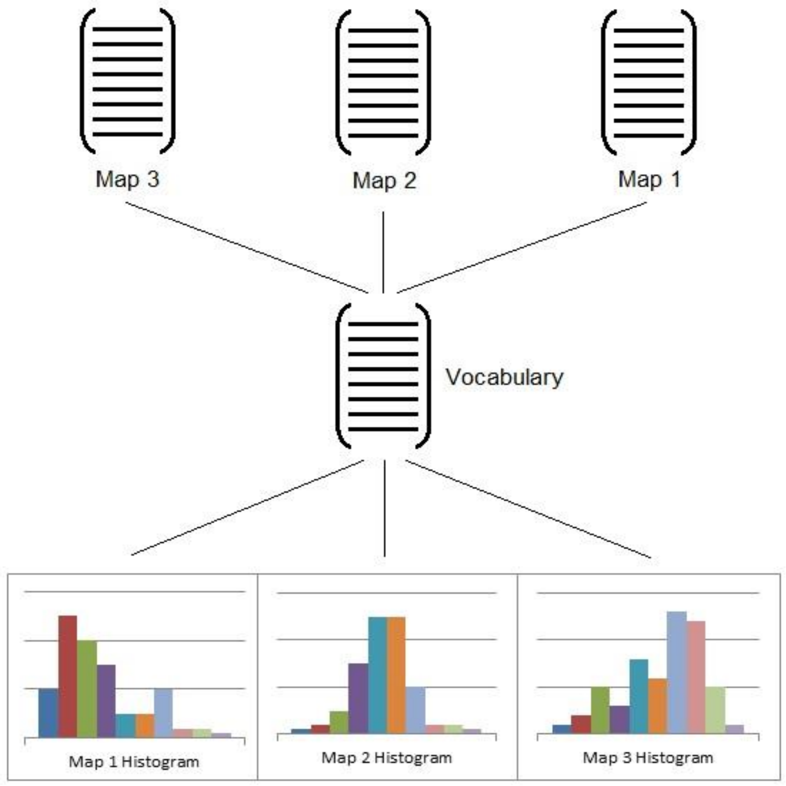

Figure 3.7 – Creating three map histograms from their descriptors and a given vocabulary.

This concludes the map creation process. At this point, a 100 word vocabulary has been created by clustering the combined set of all map descriptors, and each individual map is represented by the following data structures:

- A set of descriptors, each linked to a 3D world coordinate in the form of a clustering tree.
- A set of four 3D world coordinates denoting the map's bounding box.
- A normalized histogram obtained by matching map descriptors to the vocabulary.

3.6 AR Pipeline

The AR system operates in two possible states: *lost* mode (Figure 3.8, Section 3.6.1) and *tracking* mode (Figure 3.12, Section 3.6.2). The system is in lost mode when it is not localized to a map. While in lost mode, features are extracted and matched in every incoming camera frame. Tracking mode occurs when a map has been found with sufficient confidence and feature inliers are being tracked frame-to-frame. Both modes begin with a decoded greyscale image and result in a recovered camera pose. Lost mode is described first, followed by tracking mode.

3.6.1 Lost Mode

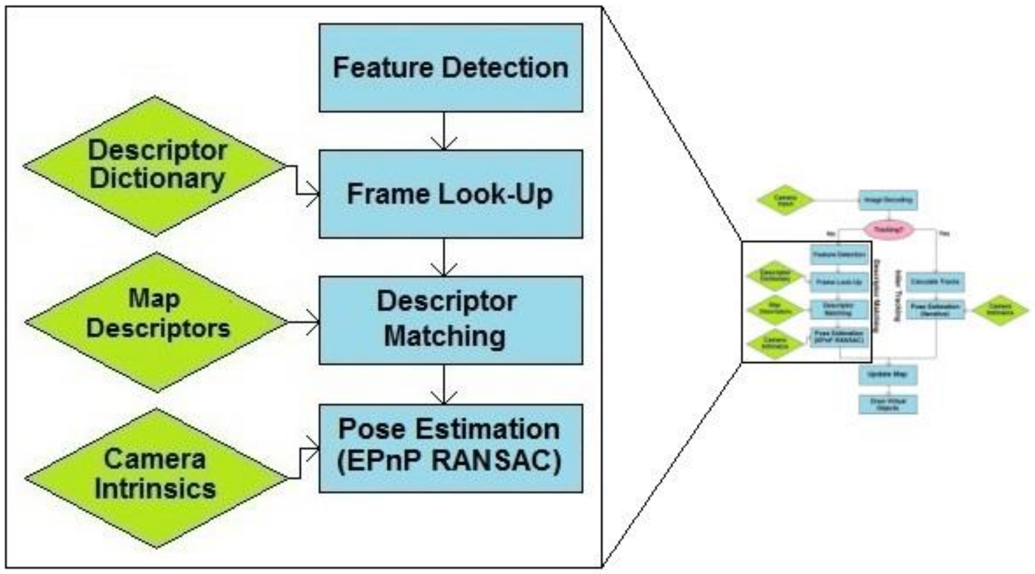

Figure 3.8 - Lost mode overview.

Lost mode begins by extracting ORB features from the decoded greyscale image. There are two cases considered when extracting features from the new camera frame. First, if the system did not find a map in the previous frame, features are extracted from the entire image and then used to determine which map, if any, the camera is looking at (Section 3.6.1.1). The second case occurs when the system found a map in a previous frame, but the number of feature inliers was insufficient to begin tracking. In this case, features are only extracted from within the bounding box of the map as it appeared in the previous frame. This bounding box is updated during the map update process (Section 3.6.3).

For the sake of processing speed, the number of feature points per camera image is limited to 300. Constraining those 300 points to lie within the region suspected to contain the map ensures that no potential match is wasted by a point lying outside the map boundary. Additionally, it is entirely possible that the region surrounding the map will be far more "interesting" (producing many interest points) than the map itself. This can cause most of the features to be found outside the map region, leaving too few within the map region to match effectively. Figure 3.9 illustrates what can happen when the number of features is limited in a camera frame containing an interesting background, as well as how extracting features only within a certain boundary solves this problem.

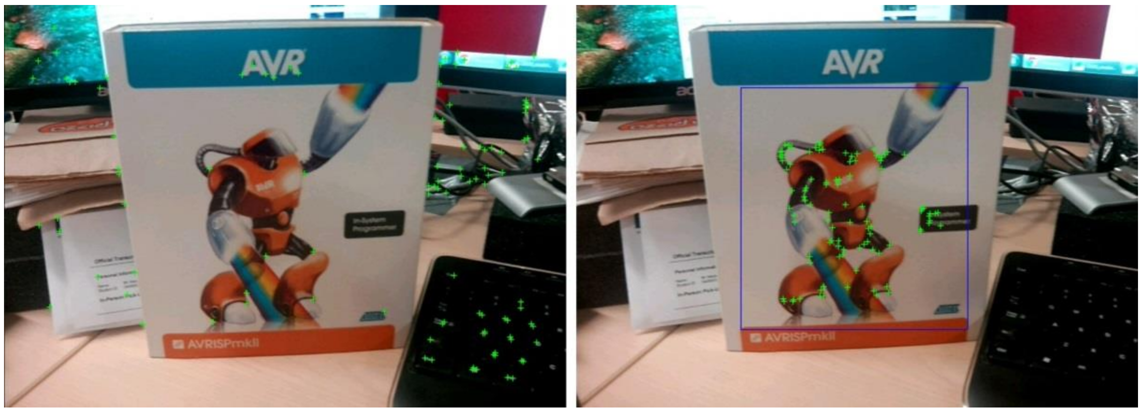

Figure 3.9 - Left: No map is found, features (green crosses) are extracted from the entire image. Right: Features are extracted only from within the map RoI (blue rectangle). For clarity, only 100 features are extracted for this example.

3.6.1.1 Localization and Matching

Once features have been extracted from the new camera frame, their descriptors are matched to the vocabulary. A normalized histogram is created in the same way as the map histograms as described in section 3.5.3. The two histograms (map and frame) are treated as 100-dimensional vectors and the Euclidean distance between them is used as a similarity measure between the camera frame and the map (Figure 3.10). The map that produces the smallest distance between histograms is considered the best match. Attempts were made to determine a suitable maximum distance threshold for whether or not a map is present in the scene, but the range of values obtained from matching histograms varied too greatly. Therefore, the best match is always chosen, even though it is possible that no map is in the scene at all. The case of no map being present is filtered out in the matching stage below.

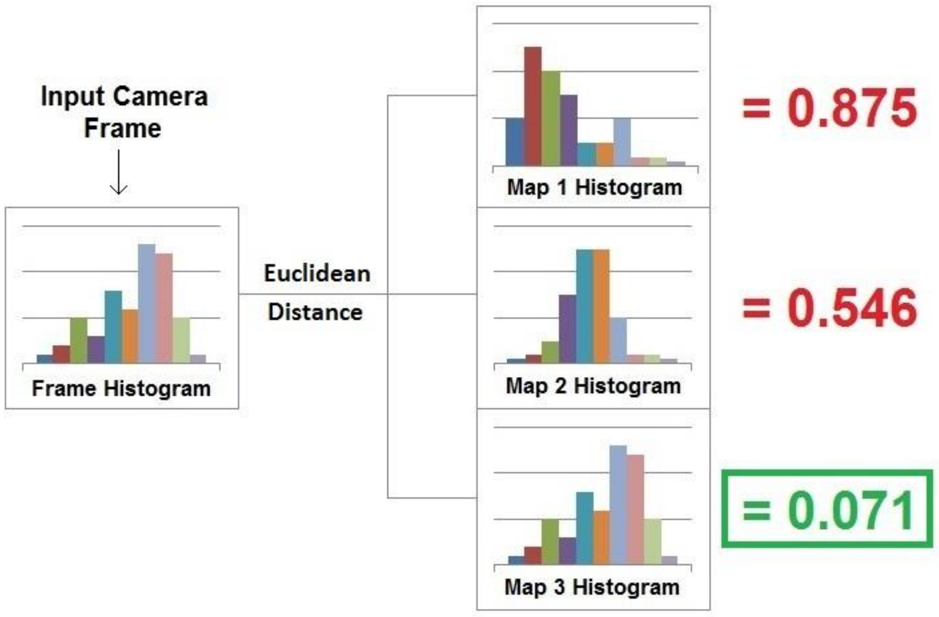

Figure 3.10 - Histogram comparison to localize to a specific map. In this case, Map 3 is the best match to the camera frame, and produces the lowest histogram distance. For clarity, 10-bin instead of 100-bin histograms are shown.

Once the best map candidate has been determined, the camera frame descriptors are matched to that map's descriptor tree using the process described in Section 3.4.6. The matching process yields a list of correspondences between 2D image coordinates (camera frame) and 3D world coordinates (map). It also provides potential starting points for the tracking phase, should the correspondences pass the subsequent pose estimation step in section 3.6.1.2 below.

3.6.1.2 Pose Estimation and Outlier Rejection

Known as the Perspective n-Point problem, it is possible to determine the position and orientation (pose) of a camera in 3D space using 2D-3D point correspondences (Figure 3.11).

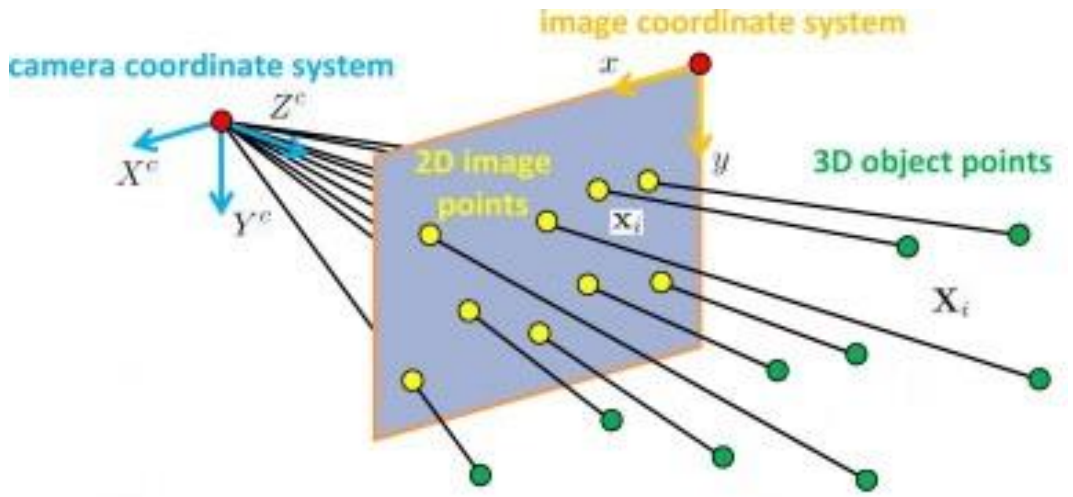

Figure 3.11 - Determining camera pose (X_c, Y_c, Z_c) from 2D-3D correspondences (x_i, X_i) (image from [54]).

Despite the limits and thresholds applied during the matching process, it is still possible for erroneous correspondences to be present. These bad matches can comprise a significant portion of the list of correspondences, and, if not rejected, degrade the accuracy of the recovered camera pose. The goal is to only use correct matches in the calculation of the camera pose. To do this, the EPnP algorithm is used inside a RANdom SAmple and Consensus (RANSAC) loop [55].

The idea behind RANSAC is to iteratively pick a minimum random subset of the data, fit a model to that subset, and then check how well that model fits the entire data set. The goal is to find the largest subset of consistent data points. In the case of determining camera pose, four random correspondences are used (four being the minimum number required by the EPnP algorithm) to compute a pose hypothesis. This hypothesis is used to build a projection matrix which can be applied to all the 3D points in the list of correspondences. Applying the projection matrix yields a list of 2D points which correspond to the 2D portion of each matching pair. If the projected 2D point is within some threshold of the matched 2D point, that match is considered an inlier for the current pose hypothesis. Once an exit condition is reached (described below), the hypothesis yielding the most inliers is chosen. Least-squares [56] pose estimation is performed using this entire set of inliers, yielding the final camera pose that best fits the set of correspondences.

The RANSAC loop requires three input parameters: (i) the maximum number of iterations, (ii) the minimum number of inliers, and (iii) the inlier threshold. The inlier threshold was set at 15.0 pixels, which is the default for OpenCV's implementation of the EPnP algorithm. This number reflects the fact that features can drift in the image by several pixels depending on the viewing conditions. Setting the value too low can result in accurate matching being rejected due to feature drift. Furthermore, even if a false match lies within this 8.0 pixel radius, it will only cause minimal damage to the integrity of the recovered pose due to the error being so small.

The maximum number of iterations was determined empirically. Since the RANSAC algorithm is not deterministic, the integrity of the camera pose depends entirely on the probability of selecting four correct correspondences within the maximum number of iterations. 5000 iterations may have a very small chance of failing to find inliers, but the running time would be far too high. Therefore, this probability is weighed against the speed of each iteration to find a middle ground between accuracy and efficiency. In practice, when the correct map is present in the camera image, the number of outliers was typically below 25%. However, under extreme conditions this number was observed to climb to 50% (or higher), so the maximum number of iterations was calculated based on a 50% outlier ratio. Additionally, the desired probability of selecting four good inliers was arbitrarily set to 99%. Equation 10 is the standard method of calculating the number of RANSAC iterations required, and can be found in [56]. The number of iterations in this case was found to be 72.

$$Iter_{MAX} = ceil\left[\frac{\log(1-R_C)}{\log(1-{R_o}^n)}\right]$$

Equation 10 - Maximum number of RANSAC iterations based on randomly selecting four inliers (n = 4) from a data set containing 50% outliers (R_o = 0.5) with a 99% confidence ratio (R_c = 0.99).

The final parameter determined was the minimum number of inliers. This simply tells the RANSAC loop to stop iterating when a hypothesis yielding over a certain number of inliers is found (even if the maximum number of iterations has not been reached). This saves time in the event of finding a suitable pose early in the iteration cycle. Since the observed outlier ratio was found to be 25% under good conditions, the minimum number of inliers was set to 75% of the matches. That is, once a hypothesis is found that fits 75% (or more) of the correspondences, the loop exits.

There is one main downside to pose estimation using feature matches: The same features are not always extracted in every frame, which means the set of inliers is always different. This causes the camera pose to differ slightly each frame, which manifests itself as jitter in the displayed augmentation. The camera may be perfectly still, but the augmentation will not be. A possible way to combat this is through pose filtering

(Kalman filter [57], Double Exponential Filter [58]). These filtering methods require a motion model, or some observed pattern within the motion to track and predict. However, due to mobile devices being largely handheld, there is always some degree of unpredictability in the motion of the device. Initial testing with double exponential filtering showed that the inherent shakiness of the device in the user's hand made it ineffective for increasing pose stability. Instead, an inlier tracking method is used.

3.6.2 Tracking Mode

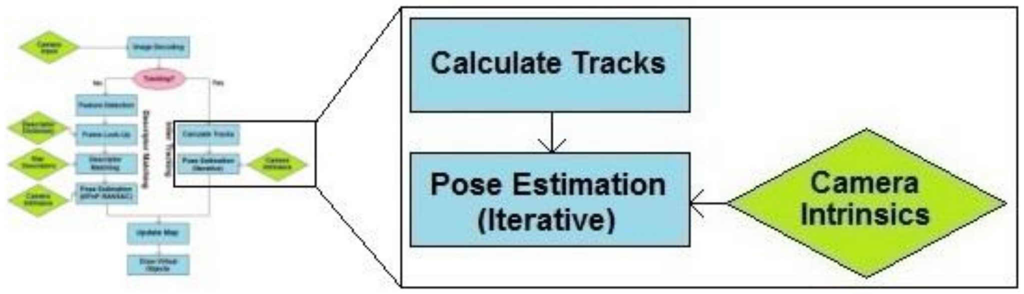

Figure 3.12 - Tracking mode overview.

Tracking mode begins when the number of inliers reaches 50% of the frame size (total number of frame descriptors). This value was obtained based on typical viewing conditions where there are 66% matches and 75% inliers, combining for 50% of the total frame size. This requires the camera to be moderately close to its original position when the map was created, but when tracking it is desirable to begin with as many inliers as possible.

When tracking begins, instead of extracting and matching new features from the current frame, inliers from the previous frame are tracked in the new frame using optic flow. This provides a number of benefits. First, tracking existing features is computationally faster than extracting new features and matching them. Second, because these are inliers, they are already known to be accurate matches. With this in mind, we make another contribution to the AR pipeline when it comes to estimating camera pose in tracking mode. Rather than performing RANSAC-based pose estimation on the tracked inliers, the system skips straight to least-squares pose estimation. This time, however, the iterative pose estimation method is used instead of EPnP. The reason for using the iterative method is that it was observed to produce more consistently robust results at extreme viewing angles. Although the iterative method is slower in general, performing least-squares iterative pose estimation is still much faster than using RANSAC-based EPnP. Using optic flow and iterative pose estimation, the system is able to track the map plane until it is nearly perpendicular to the camera plane or until it becomes occluded and still achieve a robust, stable pose.

Ideally, every inlier in the previous frame will be tracked into the new frame. This is not realistic, however. Each new frame will result in fewer and fewer successful tracks due to map occlusion, difficult viewing conditions, motion blur, or even camera noise. Eventually there will be too few tracks remaining to obtain a reliable pose. When this occurs, the system assumes the map is no longer present in the frame and reverts back to the lost mode. The lower limit on the number of tracks is set to 20. This number is based on observing augmentation stability.

3.6.3 Updating the Map

The final process in the pipeline, and another contribution made by this thesis, is to dynamically update the map. There are two aspects to this step: First, the projection matrix is used to re-project the 3D bounding corners of the map into 2D image space.

This provides the region of interest used in feature detection, and is performed in both lost and tracking modes.

The second step is to update the map with new feature descriptors. This step is only performed when enough inliers have been found to transition from lost to tracking mode. The new descriptors added are the inliers from the current frame. Each 3D point in the map can have at most two descriptors associated with it: the base descriptor obtained when the map is first created, and the most recently matched inlier descriptor. This allows features to be matched through a much wider range of motion. Since the secondary descriptor is constantly being updated, the system can continue matching to that 3D point long after the original descriptor is out of matching range. By using a secondary descriptor for each 3D point, the system achieves more robust matching under all manner of viewing conditions. Most notably, relative scale invariance can be achieved. This is useful because the ORB descriptor lacks inherent scale invariance. (Figure 3.13).

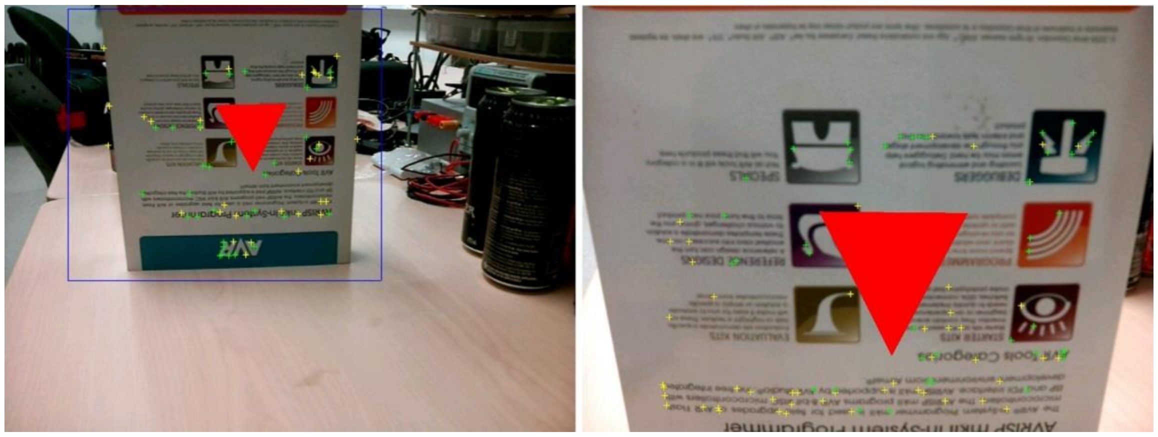
Figure 3.13 – Successfully matching to secondary descriptors while zooming in.

When using multiple descriptors for each point, an additional consideration must be made during the descriptor matching step. Since the original descriptor may be very similar to the secondary descriptor, it is very likely that together they will comprise the two best matches and fail the match ratio threshold. To prevent this, an additional condition is added to the matching process that prevents the two best matches from belonging to the same 3D point.

3.7 Drawing Augmentations

Once the map has been updated, the final task is to pass the recovered pose back to OpenGL to be used to create a model view matrix (Figure 3.14). With a model view matrix obtained via an accurate camera pose and the camera's intrinsic parameters, any 3D augmentation can be drawn. In this system, simple shapes and grids whose colour depends on the map number are drawn over the image. This thesis focused on the more challenging computer vision components of the system and has left the more trivial, but time consuming, task of more complex graphics or animations for future work.

$$M = \begin{bmatrix} R_{0,0} & -R_{1,0} & -R_{2,0} & 0.0 \\ R_{0,1} & -R_{1,1} & -R_{2,1} & 0.0 \\ R_{0,2} & -R_{1,2} & -R_{2,2} & 0.0 \\ t_{0,0} & t_{1,0} & t_{2,0} & 1.0 \end{bmatrix}$$

Figure 3.14 - Building OpenGL model-view matrix from camera pose.

This concludes the technical description of the mobile AR pipeline. For a complete reference list of all the different parameters and tuning thresholds used by each component in the system, refer to appendix A. In the next section, some sample output is shown.

3.7.1 Sample Output

Below are a series of sample output images produced by the system while tracking through different transformations as well as performing map recognition using bag of words. In Figure 3.15 to Figure 3.17, the top-left image was used to create the map, while the other three were tracked from the original map location. Figure 3.18 uses the same three map images, but this time they are all loaded into the system at once. Each of the three maps are recognized individually and then shown alongside the other two. Each map is linked to a different coloured grid, with the same solid cube being drawn in the centre regardless of which map is being tracked.

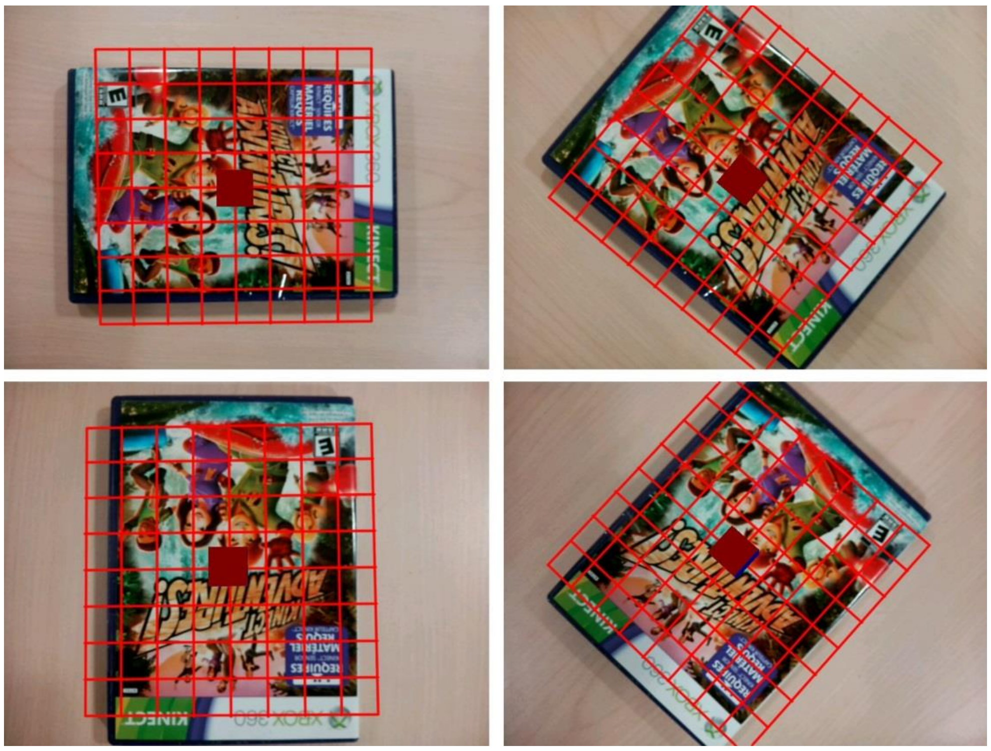

Figure 3.15 – Tracking and rendering augmentations while rotating parallel to the map plane.

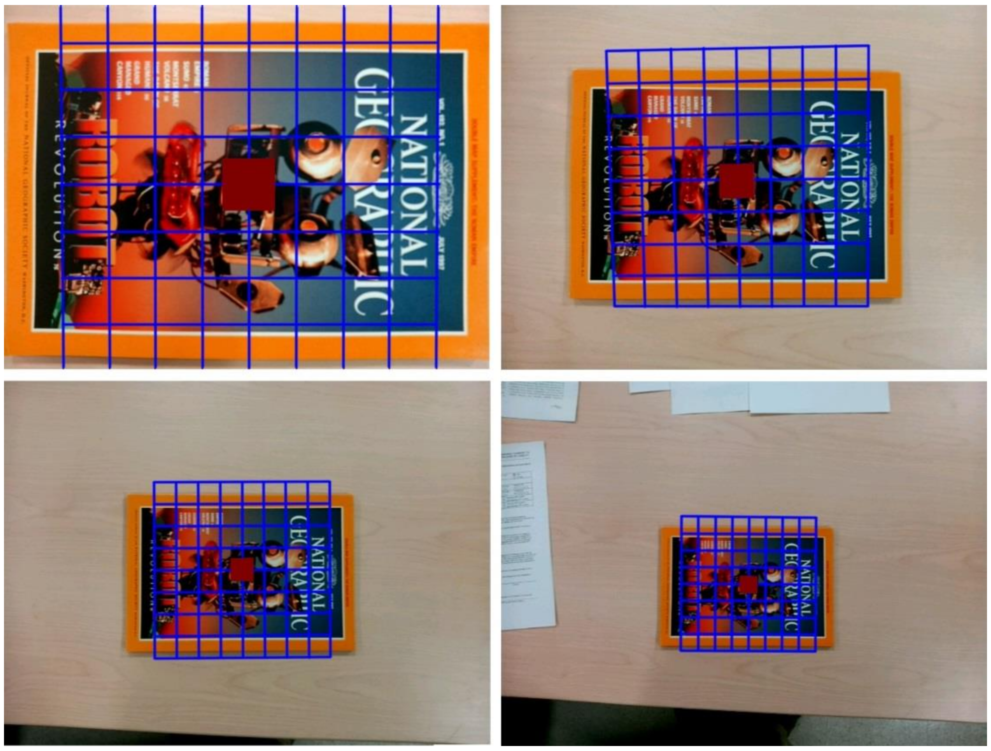

Figure 3.16 – Tracking and rendering augmentations while zooming out.

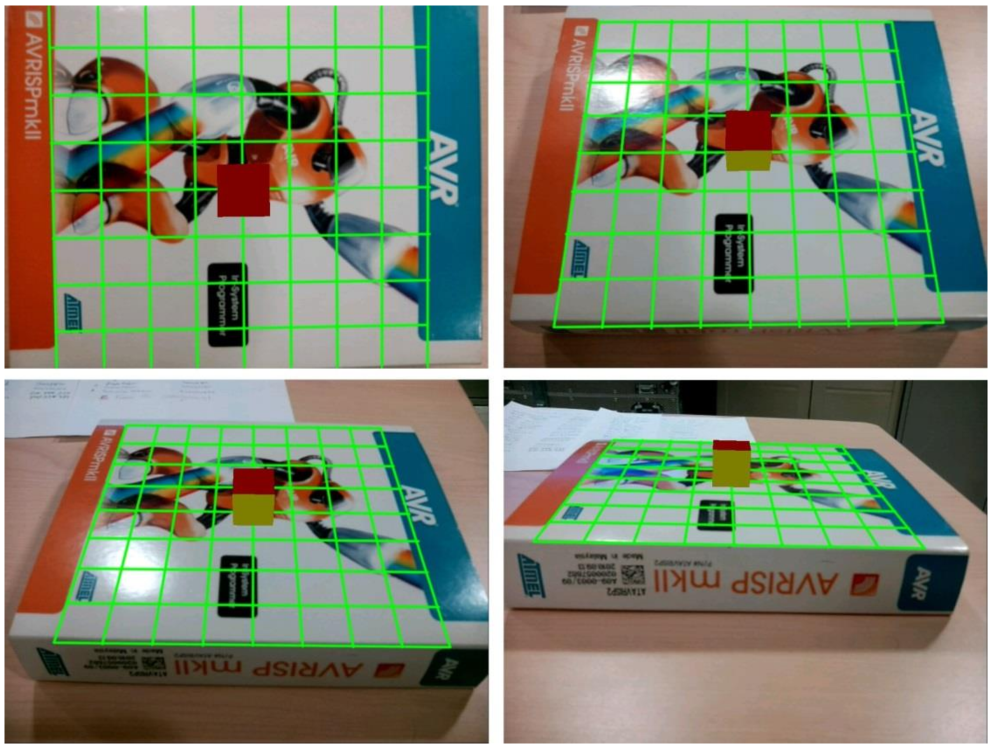
Figure 3.17 – Tracking and rendering augmentations at increasingly oblique viewpoints.

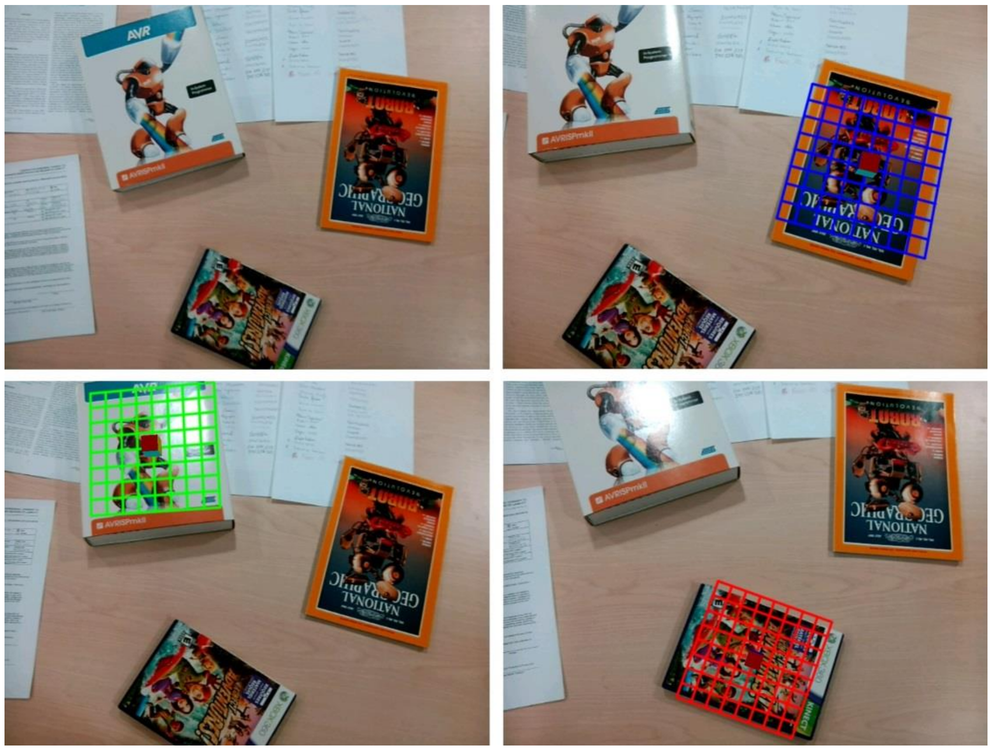
Figure 3.18 – Rendering different coloured grids depending on the map that is being tracked.

CHAPTER 4 EXPERIMENTAL RESULTS

This chapter presents the results of several experiments performed to test the system as a whole, as well as the system's individual components. Specifically, the performance of different algorithms are compared and tested for feasibility in the creation of a real time system on a mobile device. First, the data sets created for the tests are presented.

4.1 Test Data

Three still-image data sets were created to test the various components of the system. They are the "N-CART Lab" data set, the "Tabletop" data set, and the "Field Manual" data set. Each data set contains 10 reference images used as maps, and 50 query images, each observing one of the map scenes, to be used as input to the system. For the still imagery we do not include images that do not contain a map. This is because they are primarily intended to test frame lookup, which simply returns the most likely map and does not comment on whether or not a map is actually present. If there is no map present, the system will reject that frame in the descriptor matching phase and not the frame lookup phase.

In addition to the still data sets, three videos were created, one for each set. These videos are used to perform real-time tests on the entire system as if a user were actually holding the device. These videos are used in the end-to-end system tests presented in Section 4.3. All test data, including videos and still imagery, were recorded at a resolution of 640x480.

4.1.1 N-CART Lab Data Set

The N-CART set is composed of various boxes and textured objects hanging on the walls around our lab. It is designed to reflect a person walking around holding a mobile device and pointing it at the various objects. This type of data is meant to represent applications that add contextual meta-data to objects present within an environment. For example, artist information could be overlaid on paintings in a museum, directions could delivered to a user based on the visual appearance of their surroundings, or virtual points of interest could be added to an environment in real-time without having to modify the environment in any way. Figure 4.1 shows the 10 N-CART map images, and Figure 4.2 shows several sample input images from each data set. The video for this set was recorded by walking around the room recording with a handheld tablet.

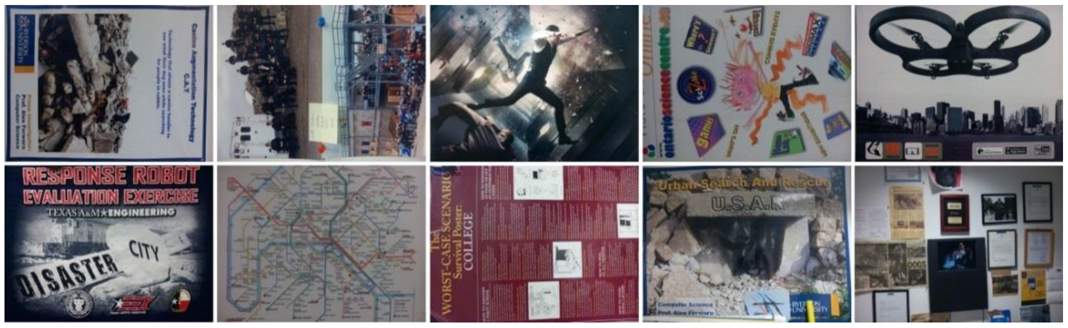
Figure 4.1 – Map images from the N-CART Lab data set.

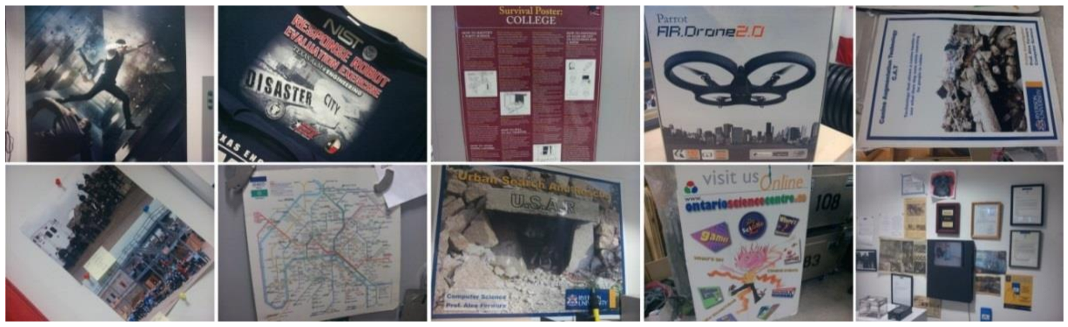
Figure 4.2 - Sample query images from the N-CART Lab data set.

4.1.2 Tabletop Data Set

The tabletop set is composed of smaller, magazine-sized sized objects with printed texture that can be placed on a desk or table. This data can be used with a handheld camera, or the camera can be stationary while the user moves the map texture around manually. Several games using special AR Tags in this manner are already on the market. This data set is designed to show that our markerless system is capable of performing this functionality as well. Figure 4.3 shows the map images from this data set, and Figure 4.4 shows several sample query images. The video for this set was taken from a top-down view using the same handheld tablet, where all the maps were laid out on a single table surface.

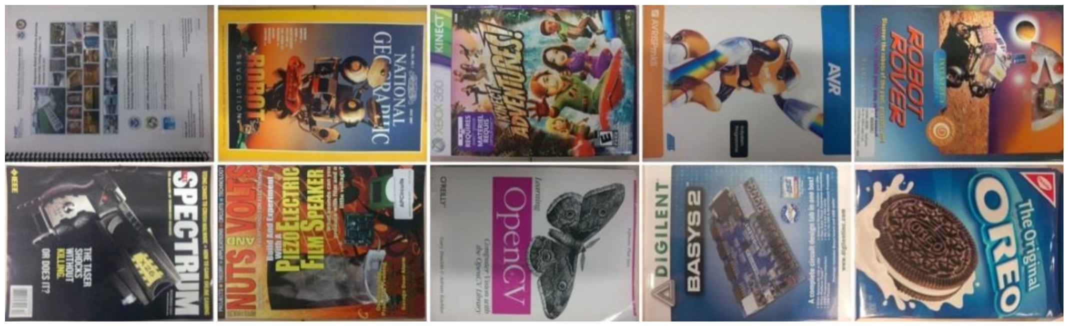
Figure 4.3 - Map images from the Tabletop data set.

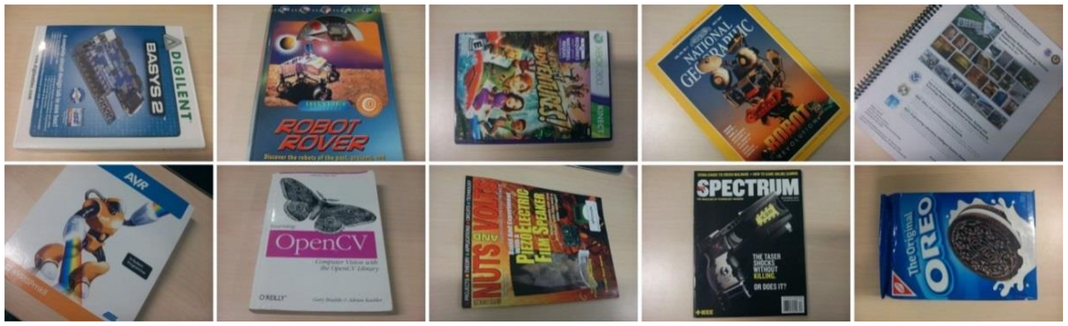
Figure 4.4 - Sample query images from the Tabletop data set.

4.1.3 Field Manual Data Set

The third and final data set is the field manual set, and is meant to be the most challenging of the three data sets. The field manual in question is a standard issue US Army Corps of Engineers Urban Search and Rescue field operations guide.

The idea behind this data set is the ability for a user to create a notebook of different virtual objects. The field manual was chosen because it is carried by all search and rescue personnel on the scene of a disaster, allowing them to quickly flip it open, create a descriptor map out of any given page and associate it with a virtual object. This is very demanding on the system because, as Figure 4.5 and Figure 4.6 show, there is very little discriminative texture on the pages of the manual. This data set challenges in particular the bag of words image recognition, but also strains descriptor matching because there are so many similar textures and patterns.

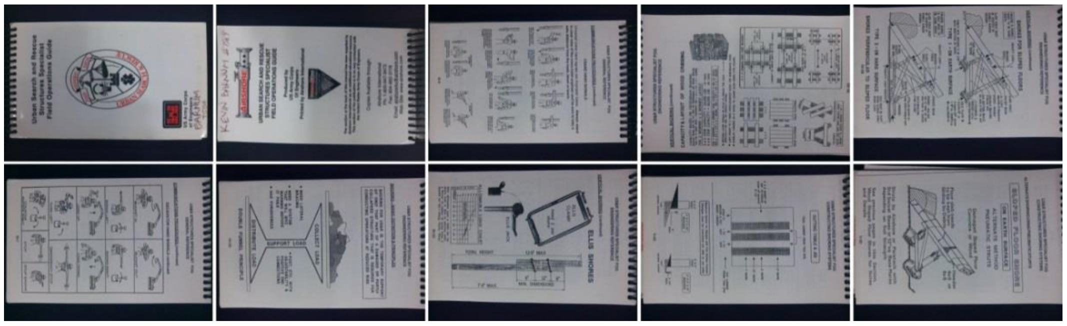

Figure 4.5 - Map images from the Field Manual data set.

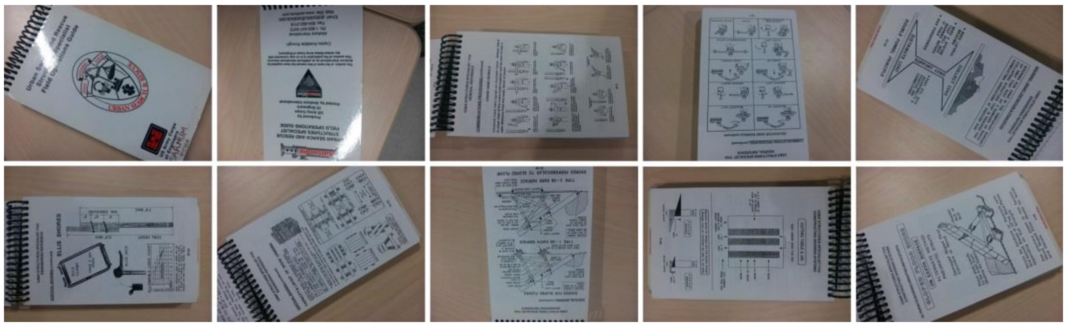

Figure 4.6 - Sample query images from the Field Manual data set.

4.2 Component Analysis

In this section, the components of the map creation process and the AR pipeline described in Chapter 3 are analyzed independently. The algorithms used by our system are compared to other candidate algorithms, with particular attention being paid to execution speed. Each component is tested on three mobile platforms: an Asus TF201 tablet (1.4 GHz), a Samsung Galaxy Tab 2 GT-P3113 (1.0 GHz), and a Samsung Galaxy Nexus GT-I9250 smartphone (1.2 GHz). Two powerful laptops are also tested: an Alienware M17 (3.3 GHz) and a Lenovo W520 (2.2 GHz). The laptop tests are conducted to illustrate the performance difference between modern computers and smartphones/tablets. No multi-threading or GPU acceleration is used in the laptop trials, just a single core of the processor.

4.2.1 Image Decoding

Image decoding speed is measured for three implementations: floating-point and integer based conversions performed on the CPU, and our NEON-accelerated integer-based conversion. Table 1 below shows the average timing results for each implementation to decode 1000 live camera image frames (640x480). The NEON architecture is not available on the laptops, so only the CPU implementations are presented. These results show over a 50% reduction in computation time across the mobile platforms when using NEON acceleration.

Platform	Clock Speed (GHz)	Average Decoding Speed (ms/frame)			NEON Speed-up
		CPU (Float)	CPU (Integer)	NEON (Integer)	
M17	3.6	5.05	**1.42**	N/A	N/A
W520	2.2	8.20	**2.81**	N/A	N/A
TF201	1.4	11.3	5.96	**2.88**	**2.07x**
GT-I9250	1.2	13.0	7.00	**3.29**	**2.13x**
GT-P3113	1.0	15.6	8.34	**3.93**	**2.12x**

Table 1 - CPU image decoding vs. NEON-accelerated decoding. Speed-up is between the CPU integer implementation and the NEON implementation.

4.2.2 Feature Detection

Speed trials were conducted to compare the chosen ORB feature detector to SIFT and SURF on each platform. In each trial, 300 features were extracted from a still image. This trial was performed 1000 times for each detector. Table 2 shows the average time required per image for each detector, and highlights a more than 10-fold speed increase achieved by using the ORB descriptor over SIFT and SURF. The efficiency of the ORB descriptor is what makes it possible to attempt markerless AR on mobile devices.

Platform	Clock Speed (GHz)	Extraction Speed (ms/frame)		
		SIFT	SURF	ORB
M17	3.6	66.5	88.1	**3.60**
W520	2.2	84.7	160	**5.87**
TF201	1.4	694	1034	**41.0**
GT-I9250	1.2	707	1154	**48.2**
GT-P3113	1.0	792	1311	**56.3**

Table 2 - Comparing detection speeds of SIFT, SURF, and ORB.

An important thing to notice about these results is how the computation time scales between the laptops and the mobile devices. While the mobile devices scale relatively evenly with each other with respect to their clock speed, the laptops are approximately an order of magnitude faster than the Asus TF201 despite having a clock speed that is only 2.6 times faster in the case of the M17, and 1.6x faster in the case of the W520.

An unexpected result of this test was that SIFT ran faster than SURF across all devices, despite SURF having been developed specifically to improve upon the slow running time of SIFT. According to the OpenCV developers, this is due to their highly efficient implementation of the SIFT algorithm [59]. Testing between this new SIFT implementation and older versions of the SIFT algorithm in OpenCV showed an approximate 5x speed-up, making it faster than SURF. Nevertheless, these trials indicate that both SURF and SIFT are unsuitable for real-time applications on mobile devices, typically requiring one second or more per frame. Figure 4.7 shows the relative scaling between the algorithms and devices.

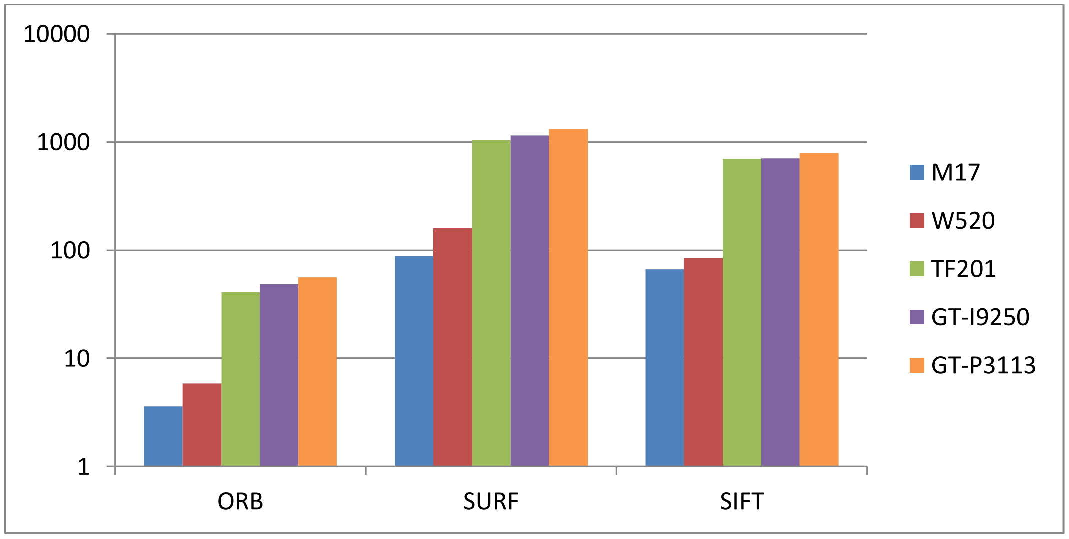

Figure 4.7 – Feature detection and description speeds (ms) of ORB, SURF, and SIFT.

4.2.3 Map Creation

Map creation involves two costly operations: extracting features (discussed above) and re-computing the vocabulary. As the number of maps increases, so does the total number of descriptors that need to be clustered to compute the vocabulary. Also, each map must then be re-matched to the new vocabulary in order to create their new response histograms. Map creation was tested using ORB, SURF, and SIFT descriptors. Figure 4.8 shows how the time required to compute a vocabulary of ORB descriptors grows linearly as the number of maps increases from one to ten. The cost of adding maps using SURF and SIFT descriptors grows linearly as well, as seen in Figure 4.9 and Figure 4.10.

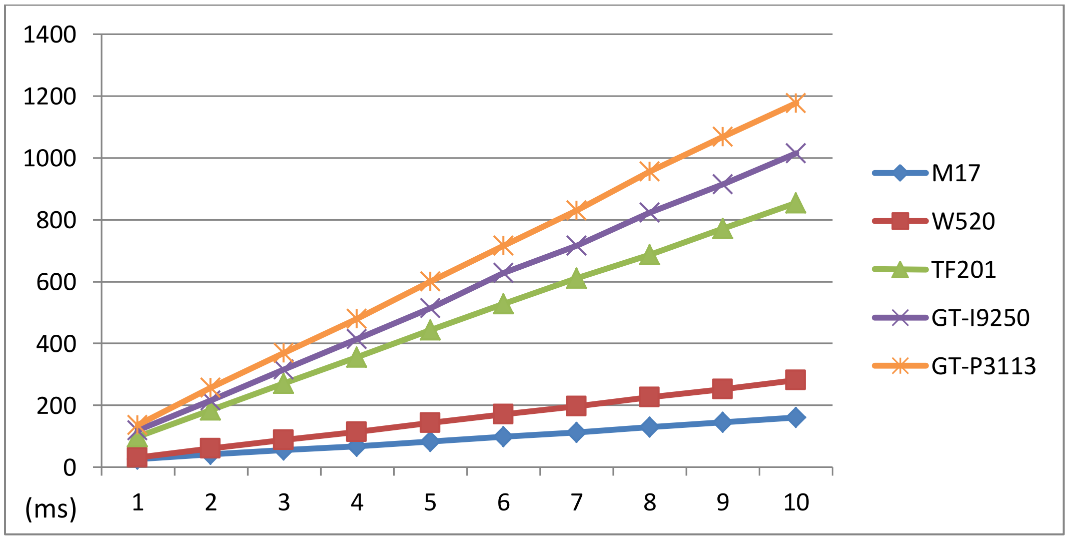

Figure 4.8 - Vocabulary computation speed (ms) using ORB (32-Byte) descriptors.

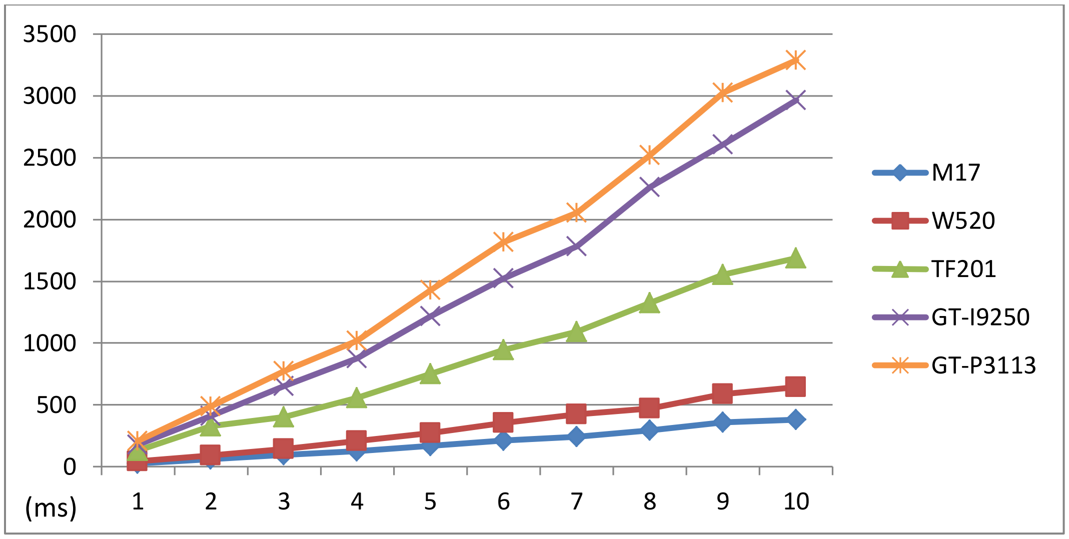

Figure 4.9 - Vocabulary computation speed (ms) using SURF (64-Float) descriptors.

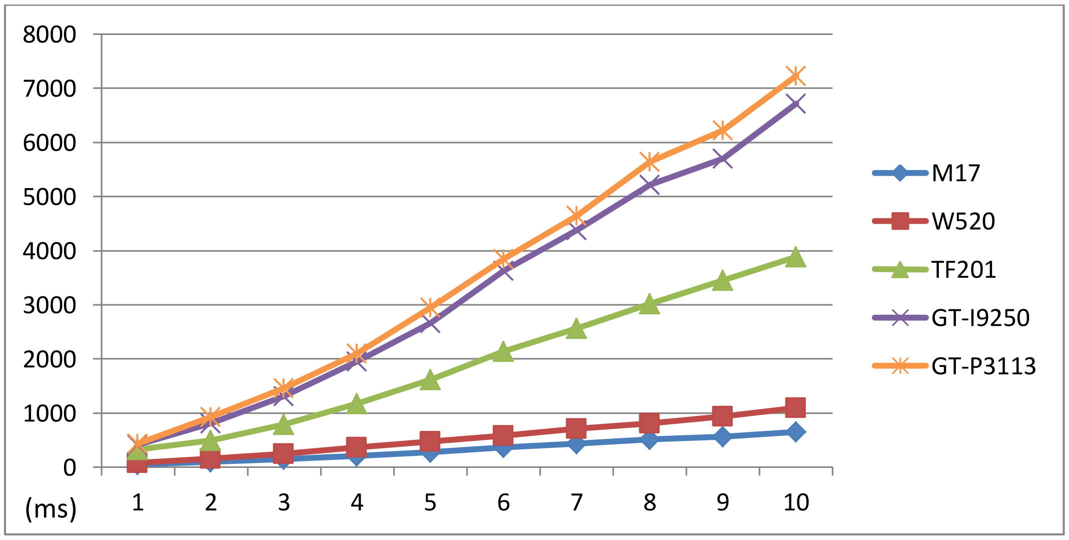

Figure 4.10 - Vocabulary computation speed (ms) using SIFT (128-Float) descriptors.

These results show that even before feature detection is considered, the cost of adding the tenth descriptor map is approximately 1.7 seconds on the TF201 in the case of SURF (64-element descriptor), and nearly four seconds in the case of SIFT (128-element descriptor). The ORB descriptor fares better, at a cost of less than one second to add the 10th map.

Because the ideal vocabulary size depends largely on the nature of the input data, this value was found empirically by testing vocabulary sizes ranging from 1 to 200 on the N-

CART lab data set. As seen in Figure 4.11, the recall rate (Equation 11) saturates very quickly; even vocabularies as small as 20 words produced recall rates approaching 80%. For our system we chose a vocabulary of 100 words, which yields a recall rate of approximately 90% while still being computationally inexpensive to compute.

$$Recall\ Rate = \frac{number\ of\ correctly\ identified\ query\ images}{number\ of\ query\ images}$$

Equation 11 – Equation to calculate recall rate.

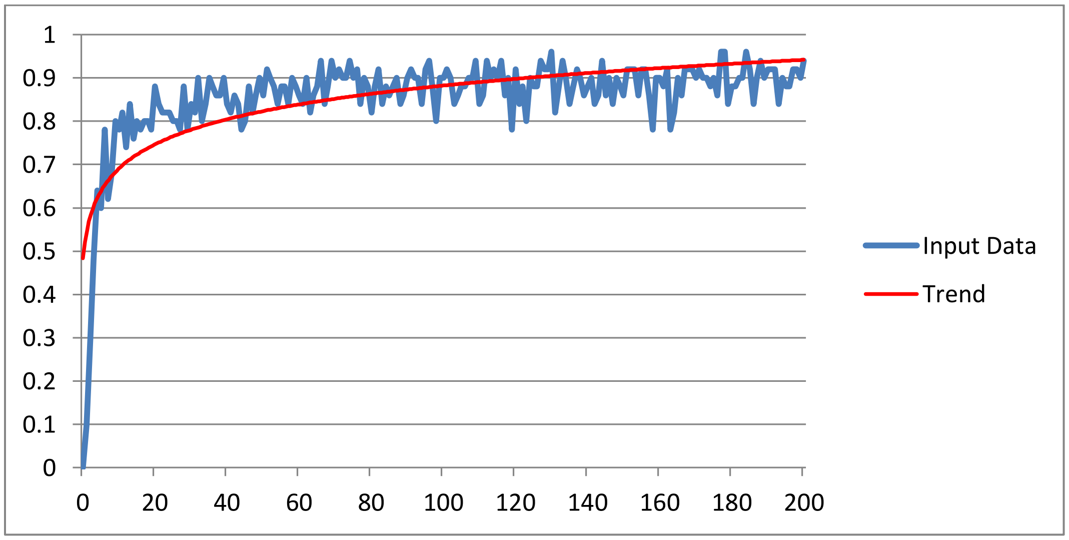

Figure 4.11 - Recall rate as vocabulary size increases.

4.2.4 Frame Look-up

This test measures the time required to match incoming frame descriptors to the vocabulary and match its histogram to the map histograms. The system was initialized with 10 maps of 500 descriptors each, and a single pre-recorded camera frame with 300 descriptors was used as input. The largest part of the computation required to look up a frame is matching its descriptors to the vocabulary and creating a histogram. Comparing two histograms is simply computing their Euclidean distance and requires a very small amount of computation. The difference between checking one map and 10 maps is negligible. Therefore the results presented here simply reflect the differences between descriptor types.

Figure 4.12 shows the average look-up speed over 1000 trials for maps for both binary and floating point descriptors. These trials do not include extracting features from the input image, just matching those features to the vocabulary and comparing histograms with each map.

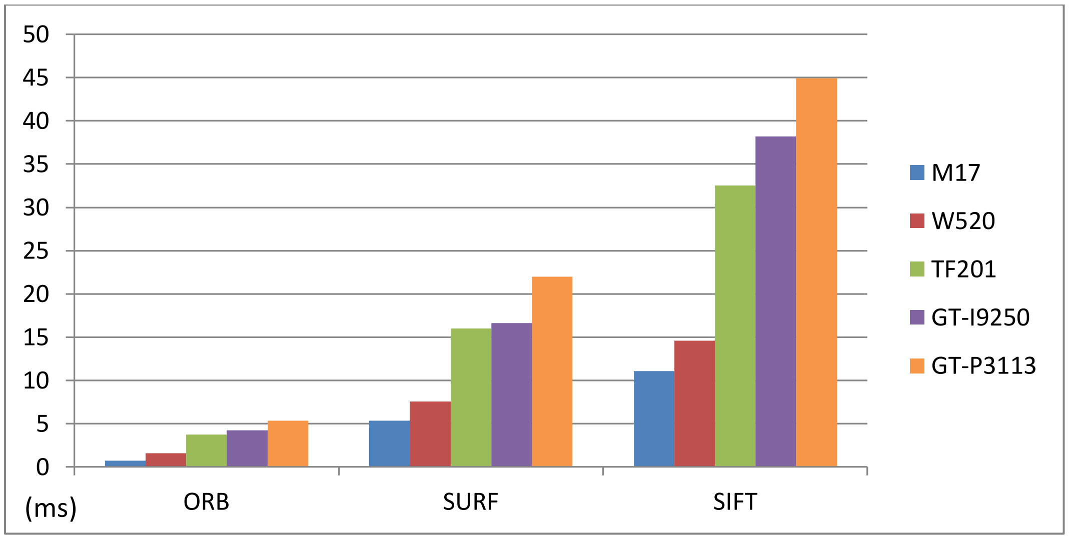

Figure 4.12 – Look-up speed (ms) averaged over 1000 frames.

4.2.4.1 Recall Accuracy

Recall accuracy is defined as how often the system returns the correct map. Again, binary ORB and floating point SIFT and SURF bag of words implementations are compared. Each descriptor type was tested on all three data sets. Recall accuracy results are shown in Table 3. Positive ID means that the system correctly identified the map in the frame, and negative ID means that the system returned a map other than the one that was actually present.

		Maps	Queries	Positive ID	Negative ID	Recall Accuracy
N-CART Lab	ORB	10	50	**46**	**4**	**92%**
	SURF	10	50	34	16	68%
	SIFT	10	50	**46**	**4**	**92%**
Tabletop	ORB	10	50	43	7	86%
	SURF	10	50	39	11	78%
	SIFT	10	50	**46**	**4**	**92%**
Field Manual	ORB	10	50	32	18	64%
	SURF	10	50	27	23	54%
	SIFT	10	50	**38**	**12**	**76%**

Table 3 - Recall accuracy for each descriptor type on all three data sets.

The above results shows that the binary bag of words implementation far exceeds the recall rate of SURF in all three data sets. It achieves results comparable to SIFT in the Lab and Tabletop data sets, while falling behind in the Field Manual data set. The binary vocabulary is several times faster to compute and has a lookup speed that is 4-8x faster across all platforms.

4.2.5 Descriptor Matching

For this trial, the comparison speed of floating point descriptors and binary descriptors are measured on each platform. These results are then compared to the NEON-accelerated implementation on the mobile platforms. Table 4 shows the comparison speed for a single pair of descriptors averaged over one million trials. Across the mobile platforms ORB descriptor comparison is approximately four times faster than SURF, and, even when using ARM's built in population count instruction, adding NEON acceleration speeds up binary descriptor comparison by an additional factor of three.

Platform	**Clock Speed (GHz)**	**Comparison Speed (μs/pair)**				**Speed-Up (ORB CPU vs. NEON)**
		128-Float (SIFT)	**64-Float (SURF)**	**256-Bit (ORB)**	**256-Bit (NEON)**	
M17	3.6	0.362	0.172	**0.0237**	N/A	N/A
W520	2.2	0.466	0.230	**0.0435**	N/A	N/A
TF201	1.4	0.886	0.455	0.132	**0.0406**	**3.25x**
GT-I9250	1.2	1.01	0.529	0.138	**0.0439**	**3.14x**
GT-P3113	1.0	1.25	0.628	0.163	**0.0565**	**2.88x**

Table 4 - Average descriptor comparison speed.

Additionally, the speed of linear search matching (brute force) is compared to the clustering tree method employed by this system. In this test, 300 frame descriptors are matched to 500 map descriptors using each method and averaged over 1000 trials. In each trial, all the matching checks and criteria outlined in section 3.4.6 are performed. Table 5 shows the results. The linear matching speed of SIFT and SURF descriptors is also measured to provide a comparison, though as with the laptop trials, they do not benefit from NEON acceleration.

Platform	**Clock Speed (GHz)**	**Matching Speed (ms/frame)**				**Speed-Up**
		Linear (SIFT)	**Linear (SURF)**	**Linear (ORB)**	**Clustering Tree (ORB)**	
M17	3.6	55.4	27.0	4.50	**0.921**	**4.89x**
W520	2.2	70.9	37.1	8.30	**1.97**	**4.21x**
TF201	1.4	180	94.3	18.8	**4.25**	**4.21x**
GT-I9250	1.2	215	111	26.2	**6.13**	**4.27x**
GT-P3113	1.0	244	129	29.4	**7.06**	**4.16x**

Table 5 - Comparing the speed of linear search matching to clustering tree matching.

4.2.6 Inlier Tracking

Since Optic Flow is independent of the detector used to initially find the tracked features, it is straightforward to measure. Table 6 shows the tracking speed for 300 features across all platforms averaged over 1000 trials.

Platform	Clock Speed (GHz)	Tracking Speed (ms)
M17	3.6	7.49
W520	2.2	13.1
TF201	1.4	22.0
GT-I9250	1.2	65.4
GT-P3113	1.0	72.6

Table 6 - Optic Flow tracking speed for 300 features.

4.2.7 Pose Estimation

Both pose estimation methods used by the system (Iterative and EPnP) were measured in terms of speed. This is straightforward for the iterative method, since it is only used in a least-squares estimation while tracking. For EPnP the computation time depends largely on the number of RANSAC iterations that are performed before a suitable pose is found. Table 7 presents speed trials for least-squares pose estimation using both the Iterative and EPnP methods to provide a baseline. In addition, a worst-case scenario of 72 RANSAC iterations is presented for EPnP. In all cases, a set of 300 correspondences is used.

Platform	Clock Speed (GHz)	Pose Estimation Speed (ms/frame)		
		Iterative	EPnP	EPnP (RANSAC)
M17	3.6	2.99	0.230	8.35
W520	2.2	3.17	0.263	9.99
TF201	1.4	10.7	1.69	20.9
GT-I9250	1.2	19.9	1.36	23.2
GT-P3113	1.0	21.0	1.57	26.4

Table 7 - Least-squares Iterative and EPnP pose estimation.

4.3 Overall Speed Evaluation

In this section, everything is put together to see whether or not the pipeline achieves the desired operational speed of 15Hz on the mobile devices. Based on the incremental results from the previous sections, Figure 4.13 shows the distribution of the major system components while tracking and Figure 4.14 to Figure 4.16 shows the same data while matching for all three feature types. The tracking figure applies to all three detector types since no features are extracted or matched while tracking.

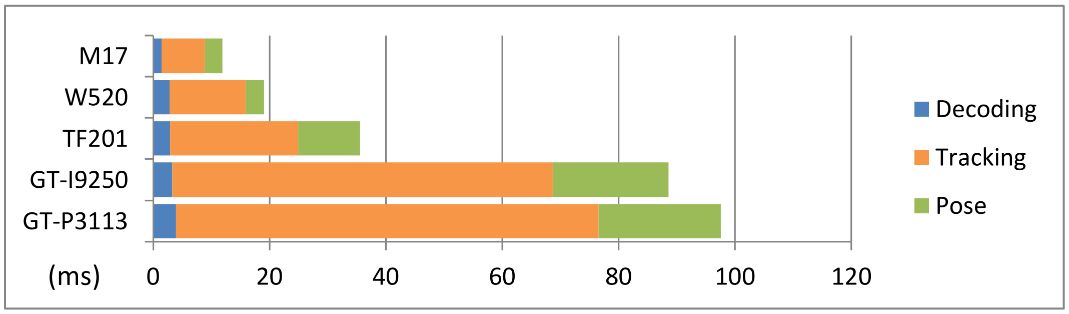

Figure 4.13 - Overall speed (ms) as a sum of the components (tracking).

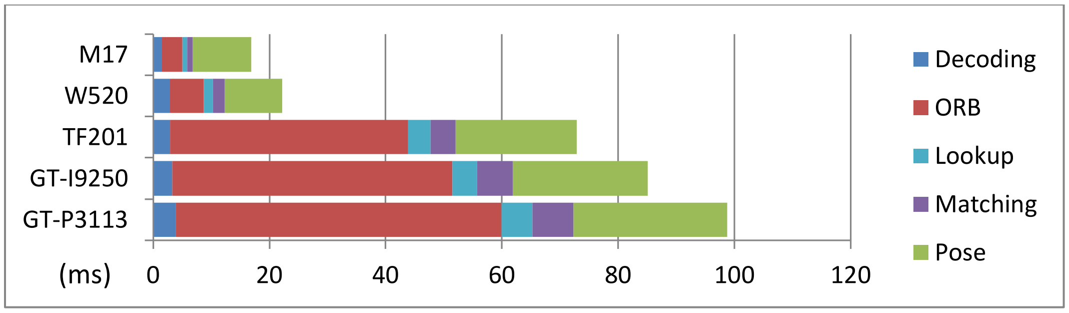

Figure 4.14 - Overall speed (ms) as a sum of the components (matching, ORB).

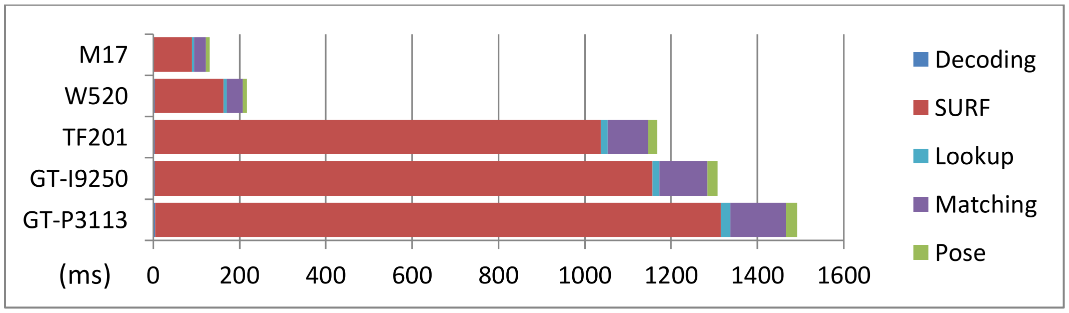

Figure 4.15 - Overall speed (ms) as a sum of the components (matching, SURF)

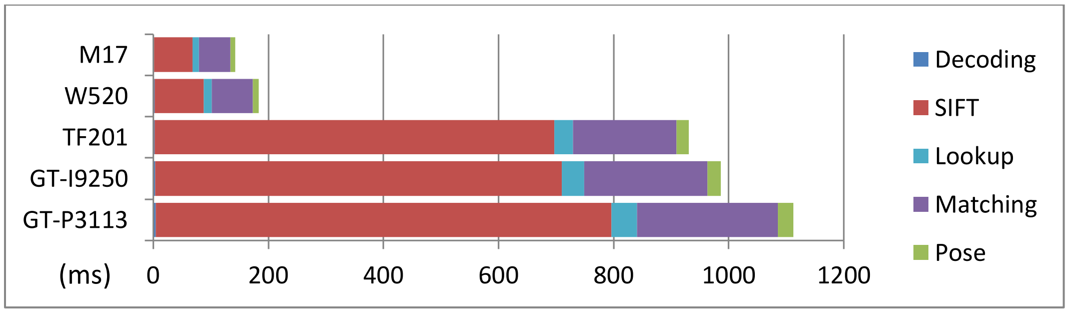

Figure 4.16 - Overall speed (ms) as a sum of the components (matching, SIFT)

This preliminary data shows that even while in tracking mode, the slower mobile devices fail to achieve the goal of 15Hz. However, each component was tested using the maximum possible number of features, correspondences, and iterations. In practice, these maximum values will almost never occur. Therefore, a series of end-to-end tests were performed on all devices with the exception of the GT-I9250. The tests were carried out on the recorded videos of each data set to more accurately reflect the real-world performance of the system. The frame-by-frame results of these tests are displayed as a moving average with a period of 30 frames in Figure 4.17 - Figure 4.19.

The speed differences between periods of tracking and matching are most clearly visible in Figure 4.17. A consistent, minimum valley is visible across the entire duration for all platforms, with peaks interspersed throughout the data. These peaks correspond to the system having lost the map and extracting and matching new features. The valleys correspond to periods of inlier tracking. In Figure 4.18 and Figure 4.19 these peaks and valleys are less defined, owing to the fact that the lab environment is far noisier than a series of objects on a plain table top, and in the field manual data set, frame lookup failed far more often. Additionally, the lab video contained a much higher percentage of frames that did not contain a map.

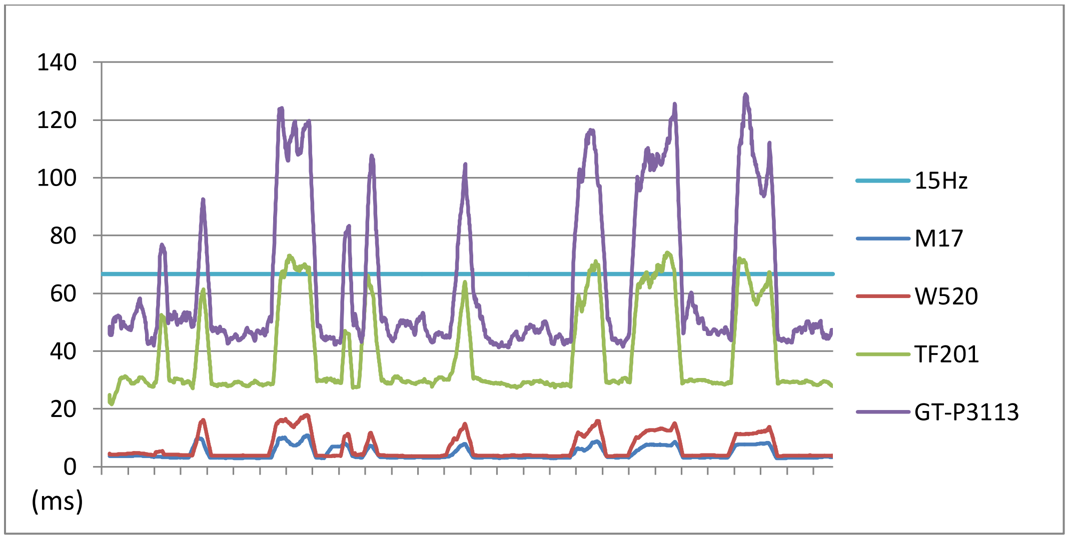

Figure 4.17 – Frame by frame speed (ms) of tabletop video (2775 frames).

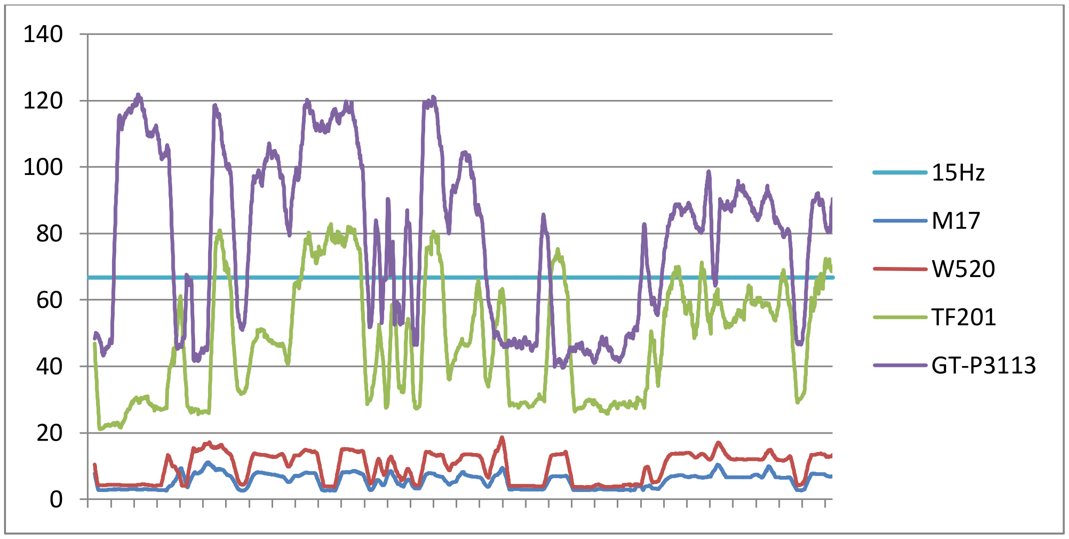

Figure 4.18 - Frame by frame speed (ms) of N-CART video (3233 frames).

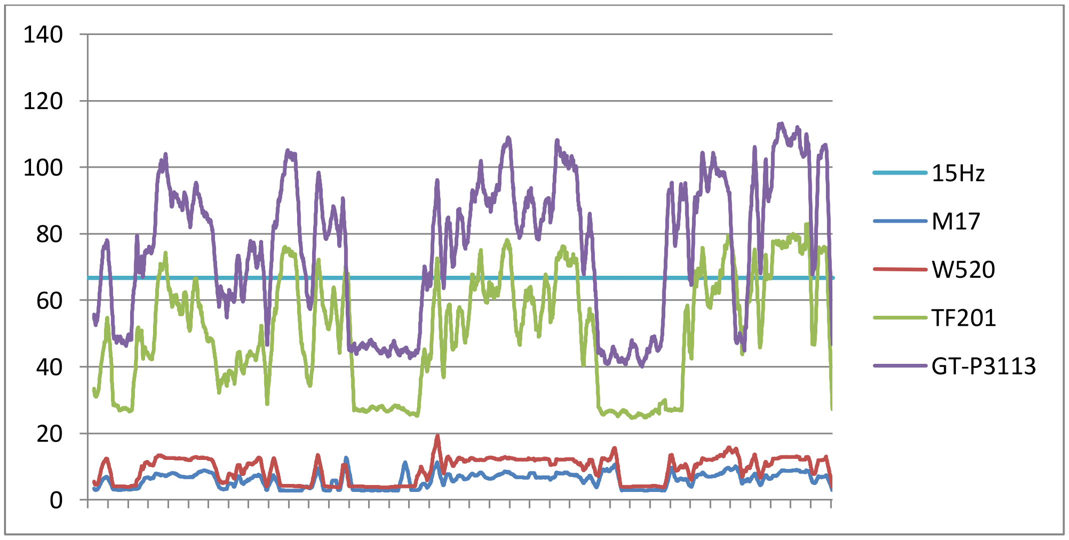

Figure 4.19 - Frame by frame speed (ms) of Field Manual video (3710 frames).

These tests show that the goal of 15Hz is achieved while tracking on all the tested devices. The TF201 very nearly achieves 15Hz while extracting features, though on the GT-P3113 frame-rates drop below 10Hz in lost mode. Slowdowns in lost mode are less critical, however, because the tracking phases directly correspond to augmentations being rendered which is when high performance is most important for the user experience.

4.4 Qualitative System Evaluation

Since the goal of any AR system is to create a seamless and immersive user experience, a series of images below show the system in action from the point of view of the user through live screen captures. Screen captures are taken from each of the three data set videos, and include several different maps being recognized and augmented. Figure 4.20 shows the N-CART lab data set, Figure 4.21 shows the tabletop data set, and Figure 4.22 shows the field manual data set.

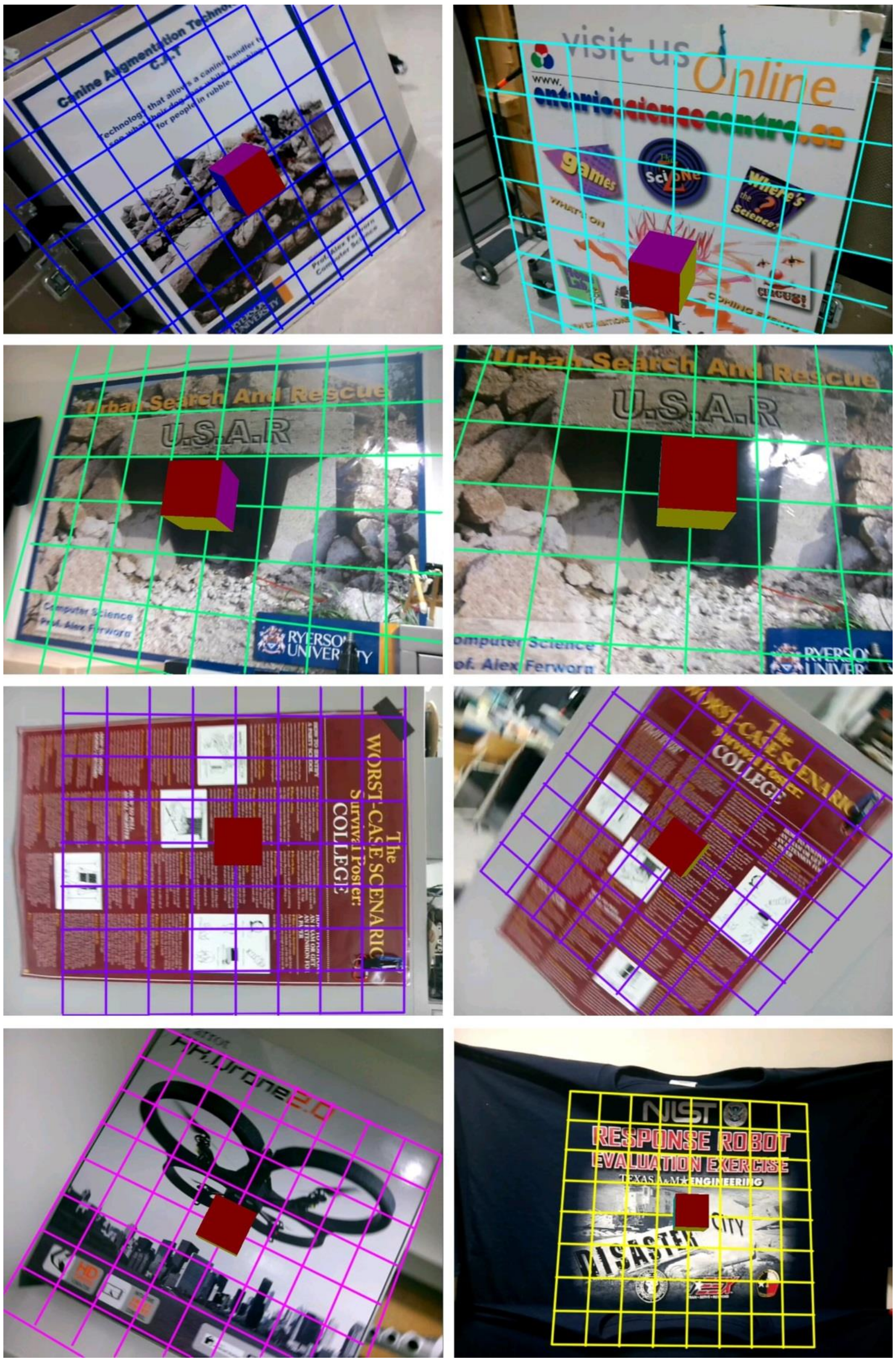

Figure 4.20 – Sample augmented imagery from the N-CART lab data set.

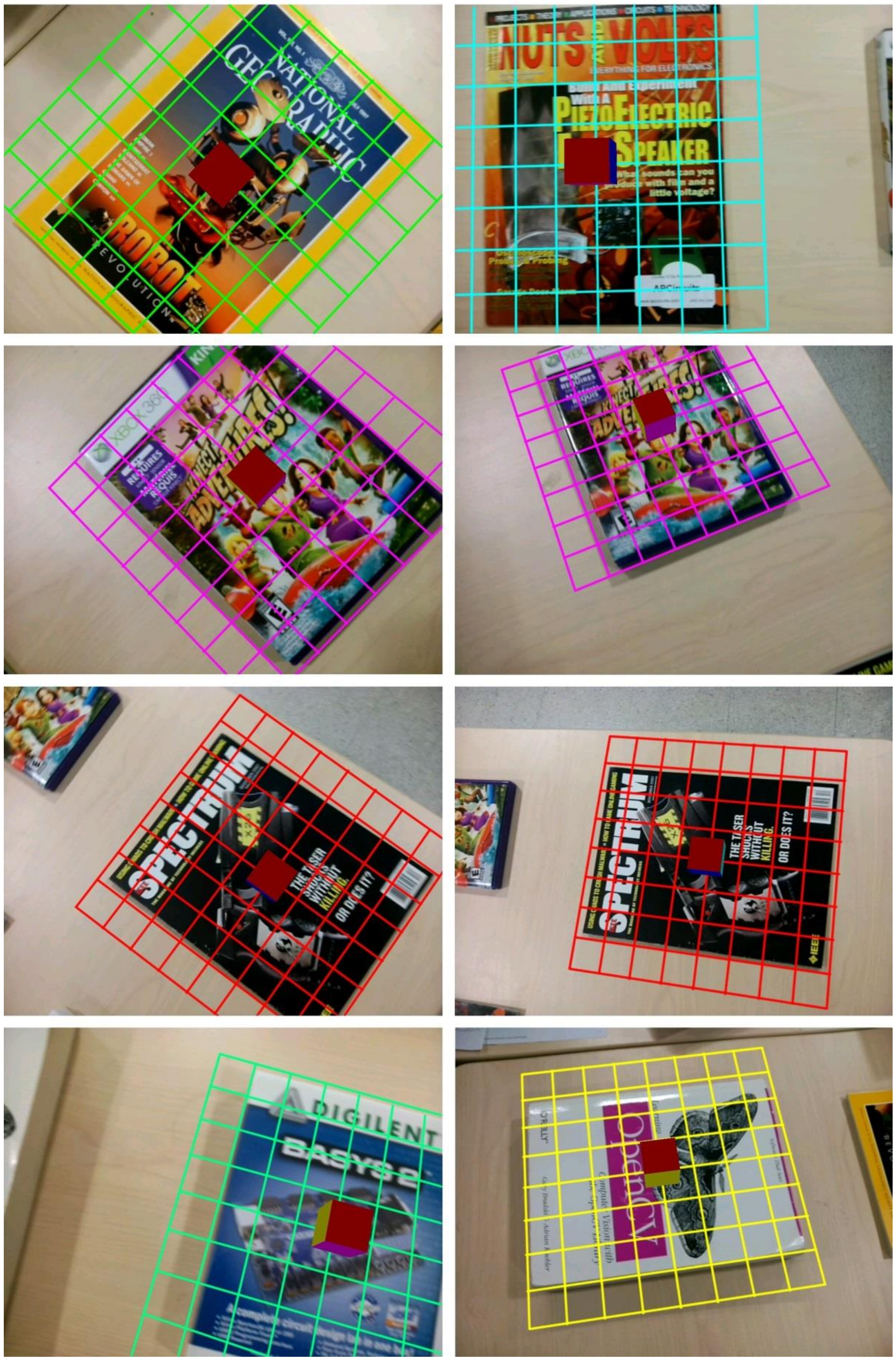

Figure 4.21 - Sample augmented imagery from the tabletop data set.

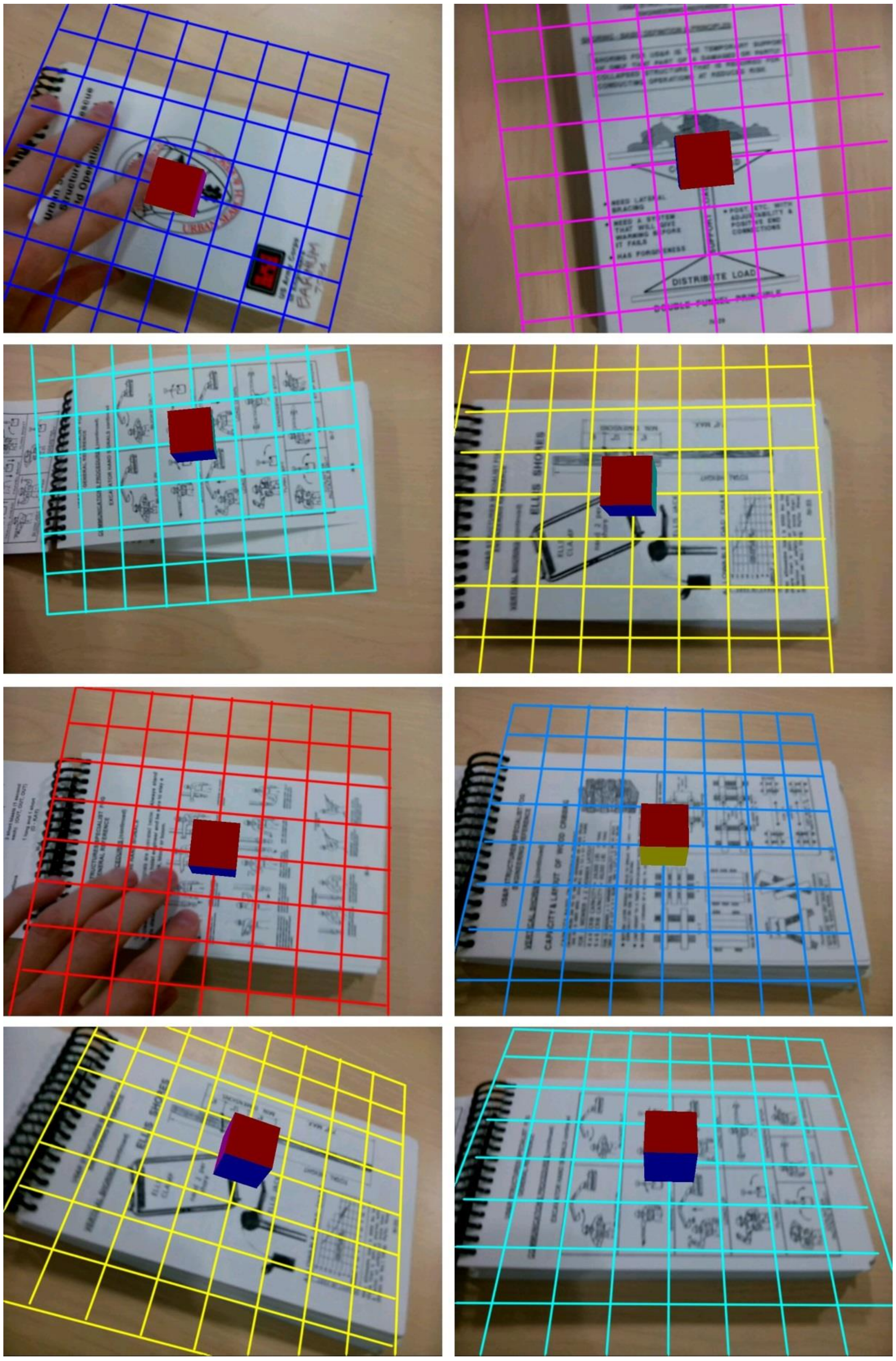

Figure 4.22 - Sample augmented imagery from the field manual data set.

CHAPTER 5 CONCLUSION AND FUTURE WORK

5.1 Summary of Results

In this thesis we presented a markerless augmented reality system designed to operate at 15Hz on consumer grade mobile devices. Numerous modifications and improvements were made in order to adapt the traditional markerless AR pipeline to the mobile domain.

Chapter 1 began with an introduction to augmented reality and outlined the challenges faced when developing an augmented reality system. The problems facing the development of our particular system were characterized, and our contributions to the body of research were outlined.

In Chapter 2 a literature review of all the candidate algorithms considered for use in our pipeline was provided. Most importantly, we analyzed several different feature detectors and descriptors to determine which would be best suited for use on mobile hardware. Several complete AR systems were also reviewed, including those also implemented for mobile devices.

Chapter 3 documented the implementation of our system. At the core of our AR pipeline is the binary ORB descriptor. It was shown that ORB descriptors can be extracted and matched in real-time on mobile devices due to the high efficiency of binary arithmetic. However, we made several additional improvements to the feature detection and matching processes. First, through the use of a dynamically updated secondary ORB descriptor we achieved a level of scale invariance without the use of costly image pyramids. This method adds very little computation overhead and was found to aid in scene recognition from outside the home location of where the descriptor map was made.

Secondly, we achieved a threefold increase in descriptor comparison speed through a SIMD implementation of the Hamming distance calculation. Finally, by storing descriptor maps in the form of a clustering tree, we achieved better than a fourfold increase in matching speed.

Improvements were also made to the image decoding step. Our SIMD implementation used a reduced integer-based conversion that allowed twice as many pixels as usual to be processed simultaneously. This resulted in an approximate twofold speed-up over a typical CPU implementation.

Our system also made use of an adaptive pose estimation scheme to balance efficiency with accuracy. By forgoing a RANSAC loop while in tracking mode and instead relying on optic flow to produce an accurate list of inliers, we greatly improved the speed of pose estimation while at the same time eliminating the vast majority of augmentation jitter.

Our final contribution was the development of a binary bag of words scene recognition algorithm that can be applied online and in real-time. Using a novel approach to computing a binary descriptor vocabulary using k-medians, we showed that at least 10 unique descriptor maps can be added to the system while achieving a recall rate of approximately 90%. Furthermore, we showed that these results are comparable to and in some cases exceed those of equivalent implementations based on floating-point descriptors while requiring a fraction of the computation time.

In Chapter 4, numerous experiments were carried out on three mobile devices of varying hardware capability using three unique data sets. The system was shown to operate at speeds of 10-30Hz (depending on the device) while rendering accurate and stable augmentations. This validated our goal of 15Hz laid out at the beginning of this thesis and proved our hypothesis that robust markerless AR is possible on current generation mobile devices.

Finally, the contributions and results presented in this thesis were accepted as a conference paper at the 10th Conference on Computer and Robot Vision (CRV 2013) [60].

5.1.1 Limitations

Despite having achieved the goals we set for this system, there are a few limitations that were identified. The most apparent of these is that the system depends on there being sufficient unique texture in the map images to provide an acceptable number of salient feature points. Scenes such as blank walls or those containing parallel lines (e.g. wooden flooring) do not provide detectable interest points. Also, scenes with repetitive textures such as a tiled floor or a brick wall may provide enough interest points, but very few will be uniquely matchable. Possible ways to accommodate these real world conditions are sensor fusion approaches using onboard motion and orientation sensors.

Second is the requirement that maps be planar, which is a very common limitation for AR systems. In our case, it is a constraint used to simplify the map creation process. If the system were provided a-priori with a non-planar list of 2D-3D correspondences to be used as a map, our system could accommodate it without modification. The clustering tree and bag of features vocabulary could be created, pose could be estimated and augmentations drawn without any changes to the pipeline.

Another limitation is that maps must be viewed at or near the location they were created to be reliably recognized by the bag of features algorithm. This limitation is a result of different ORB features being detected depending on the viewpoint of the camera. If the camera is in a similar position as when the map was created (parallel to camera plane, similar distance) most of the same map features will also be extracted from the camera frame. Otherwise, the features detected in the camera frame will differ too greatly from the map and result in poorly matching histograms. This is a problem with using an untrained bag of features scheme. It is solvable by supplying training data in the form of multiple map images at different viewpoints, but this goes against our concept of a simplified, one-click map creation process.

5.2 Future Work

In the absence of GPU acceleration, our system used ARM's SIMD engine to decode images and perform descriptor comparison. Future work will involve implementing feature detection and descriptor extraction using NEON and the SIMD engine. If the system's other NEON implementations are any indication, this will greatly improve the worst-case execution speed of the system and perhaps allow feature matching to be performed on top of optic flow tracking to provide a more robust pose estimate.

In addition, as hardware evolves, acceleration methods used in PC implementations will become increasingly relevant to mobile hardware. A popular way to improve algorithm efficiency is to parallelize through GPU acceleration. To the best of our knowledge, as of this writing, there are no consumer-grade mobile devices on the market that contain CUDA-programmable GPUs. However, Nvidia recently unveiled their new Tegra 4 processor, which will contain a programmable 72-CUDA core GPU [61]. Additionally, OpenCL (Open Computing Language) [62] is becoming more and more prevalent for general purpose programming on mobile GPUs, though the performance increases are not as visible as their full-sized counterparts. Several simple demos including contour tracking and image convolution can be found at [63]. Most of the algorithms used in our AR pipeline can benefit greatly from parallelization, including image decoding, descriptor extraction, pose estimation, descriptor matching, and frame lookup. As programmable GPUs become widely available on mobile devices, not only can current algorithms be implemented more efficiently, but fewer trade-offs will have to be made in the name of computation speed.

As efficiency increases, additional functionality can be added. Currently, the system cannot identify multiple maps in a single camera frame. It will just pick whichever of the two returned a better lookup score. Future work will involve adding functionality to recognize multiple maps in a scene and render unique augmentations for each map. This could be done by accepting multiple maps whose look-up scores are below some threshold, or within a certain percentage of one another. Once potential maps have been selected, the descriptor matching and pose estimation processes would be applied independently for each one. The matching process is fast enough through NEON acceleration that it could be performed multiple times without a large impact on performance. Though pose estimation is much slower, the effect of doing it more than once could be absorbed by having sped up feature detection using NEON as mentioned previously. This would yield multiple camera poses, each of which would be sent back to OpenGL where separate augmentations would be drawn.

We would like to improve the map creation process to allow for panoramic maps to be created from several overlapping images of a continuous flat surface. This could be done using image stitching techniques, similar to the way existing panorama apps work.

This would allow for the creation of large, seamless map spaces with consistent coordinate systems that would otherwise be comprised of multiple discrete maps.

5.3 Final Remarks

It was the intention of this thesis to demonstrate that markerless augmented reality can be achieved on current generation mobile devices. Looking forward, it is our hope that this work will provide the foundation and motivation for markerless AR to leave the lab and enter the hands of a more general user base. Markerless AR performed on widely available consumer devices provides an exciting opportunity to make AR applications usable by anyone, at any time.

APPENDICES

A. List of Constants and input parameters

MAP CREATION		
Category	**Name**	**Value**
Feature Detection	Feature Count	500
	Scale Factor	1.0
	Levels	1
	Edge Threshold	21
	Patch Size	21 px
Descriptor Tree	Branch Count	8
	Map Depth	5.0 m
Vocabulary Building	Vocabulary Size	100
	k-Medians Iteration Count	10
	k-Medians Exit Threshold	10 bits
Other		

AR PIPELINE		
Category	**Name**	**Value**
Feature Detection	Feature Count	300
	All Else	Same as map creation
Descriptor Matching	Hamming Threshold	70 bits
	Ratio Threshold	0.7
	Minimum Match Count	20
Inlier Tracking	Minimum Inlier Count	20
	Maximum Track Error	8.0 px
	Search Window	20x20 px
Pose Estimation	Iteration Count	72
	Exit Threshold	0.75*numMatches
	Re-projection Threshold	15.0 px

BIBLIOGRAPHY

[1] J. Woodwark, "Augmented Reality," *Computer-Aided Design*, vol. 25, pp. 466-467, 1993.

[2] P. D. Lever, "A Photomontage System for Site Planning," *Computer-Aided Design,* vol. 5, pp. 103-104, 1973.

[3] W. Piekarski and B. Thomas, "ARQuake: The Outdoor Augmented Reality Gaming System," *Communications of the ACM - Internet Abuse in the Workplace and Game Engines in Scientific Research,* vol. 45, pp. 36-38, 2002.

[4] Nintendo. (2011). *Nintendo 3DS Built-In Software*. Available: http://www.nintendo.com/3ds/built-in-software#/4

[5] J. Fritz, U. Paweena, T. Ungi, A. J. Flammang, G. Fichtinger, I. I. Iordachita, and J. A. Carrino, "Augmented Reality Visualization with Use of Image Overlay Technology for MR Imaging–guided Interventions: Assessment of Performance in Cadaveric Shoulder and Hip Arthrography at 1.5 T," *Radiology,* vol. 265, pp. 254-259, 2012.

[6] F. K. Wacker, S. Vogt, A. Khamene, J. A. Jesberger, S. G. Nour, D. R. Elgort, F. Sauer, J. L. Duerk, and J. S. Lewin, "An Augmented Reality System for MR Image–Guided Needle Biopsy: Initial Results in a Swine Model," *Radiology,* vol. 238, pp. 497-504, 2006.

[7] BMW. (2009). *2009 BMW 7 Series Head-Up Display*. Available: http://www.bmw.ca/ca/en/insights/technology/bmw_connected_drive/head_up_display.html

[8] Qualcomm. (2012). *House Model AR*. Available: https://play.google.com/store/apps/details?id=org.monosock.shadowdemo

[9] J. Gausemeier, J. Fruend, C. Matysczok, B. Bruederlin, and D. Beier, "Development of a Real Time Image Based Object Recognition Method for Mobile AR Devices," in *International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*, 2003, pp. 133-139.

[10] M. Fiala, "ARTag, A Fiducial Marker System Using Digital Techniques," in *Computer Vision and Pattern Recognition*, 2005, pp. 590-596.

[11] H. Kato. (1999). *ARToolKit Project*. Available: http://www.hitl.washington.edu/artoolkit/

[12] D. C. C. Tam and M. Fiala, "A Real-Time Augmented Reality System Using GPU Acceleration," in *Canadian Conference on Computer and Robot Vision*, Toronto, Ontario, 2012, pp. 101-108.

[13] M. Fiala and G. Roth, "Magic Lens Augmented Reality: Table-top and Augmentorium," in *Special Interest Group on Graphics and Interactive Techniques*, San Diego, California, 2007, p. 152.

[14] G. Klein and D. Murray, "Parallel Tracking and Mapping on a Camera Phone," in *IEEE International Symposium on Mixed and Augmented Reality*, 2009, pp. 83-86.

[15] G. Klein and D. Murray, "Parallel Tracking and Mapping for Small AR Workspaces," in *International Symposium on Mixed and Augmented Reality*, 2007, pp. 225-234.

[16] C. Harris and M. Stephens, "A Combined Corner and Edge Detector," in *Alvey Vision Conference*, 1988, pp. 147-151.

[17] Njw000. (2010). *Intensity Image with Gradient Images*. Available: http://en.wikipedia.org/wiki/File:Intensity_image_with_gradient_images.png

[18] S. Jianbo and C. Tomasi, "Good Features to Track," in *Conference on Computer Vision and Pattern Recognition*, 1994, pp. 593-600.

[19] E. Rosten and T. Drummond, "Machine Learning for High-Speed Corner Detection," in *European Conference on Computer Vision*, 2006, pp. 430-443.

[20] E. Rosten and T. Drummond, "Fusing Points and Lines for High Performance Tracking," in *International Conference on Computer Vision*, 2005, pp. 1508-1515.

[21] M. Fiala, "CPS843 Course Notes," Ryerson University2010.

[22] B. Horn and B. G. Schunck, "Determining Optical Flow," *Artificial Intelligence*, vol. 17, pp. 185-203, 1981.

[23] C. Tomasi and T. Kanade, "Detection and Tracking of Point Features," Carnegie Mellon University1991.

[24] B. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," in *International Joint Conference on Artificial intelligence*, Vancouver, BC, Canada, 1981, pp. 674-679.

[25] D. G. Lowe, "Object Recognition from Local Scale-Invariant Features," in *International Conference on Computer Vision*, 1999, pp. 1150-1157 vol.2.

[26] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded-Up Robust Features," in *European Conference on Computer Vision*, 2006, pp. 404-417.

[27] P. Furgale and C. H. Tong. (2010). *Speeded-Up SURF*. Available: http://asrl.utias.utoronto.ca/code/gpusurf/

[28] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary Robust Independent Elementary Features," in *European Conference on Computer Vision*, 2010, pp. 778-792.

[29] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An Efficient Alternative to SIFT or SURF," in *IEEE International Conference on Computer Vision* 2011, pp. 2564-2571.

[30] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg, "Pose Tracking from Natural Features on Mobile Phones," in *International Symposium on Mixed and Augmented Reality*, 2008, pp. 125-134.

[31] W.-C. Chen, Y. Xiong, J. Gao, N. Gelfand, and R. Grzeszczuk, "Efficient Extraction of Robust Image Features on Mobile Devices," presented at the International Symposium on Mixed and Augmented Reality, 2007.

[32] M. Muja and D. G. Lowe, "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration," in *International Conference on Computer Vision Theory and Applications*, 2009, pp. 331-340.

[33] M. Muja and D. G. Lowe, "Fast Matching of Binary Features," in *Canadian Conference on Computer and Robot Vision*, 2012, pp. 404-410.

[34] Z. S. Harris, "Distributional Structure," *Word*, vol. 10, pp. 146-162, 1954.

[35] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual Categorization with Bags of Keypoints," in *European Conference on Computer Vision*, 2004, pp. 1-22.

[36] J. Winn, A. Criminisi, and T. Minka, "Object Categorization by Learned Universal Visual Dictionary," in *International Conference on Computer Vision*, 2005, pp. 1800-1807.

[37] D. Galvez-Lopez and J. D. Tardos, "Real-time loop detection with bags of binary words," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, 2011, pp. 51-58.

[38] D. Oberkampf, D. F. DeMenthon, and L. S. Davis, "Iterative Pose Estimation Using Coplanar Points," in *Conference on Computer Vision and Pattern Recognition* 1993, pp. 626-627.

[39] H. Araújo, R. L. Carceroni, and C. M. Brown, "A Fully Projective Formulation to Improve the Accuracy of Lowe's Pose-Estimation Algorithm," *Computer Vision and Image Understanding,* vol. 70, pp. 227-238, 1998.

[40] D. Grest, T. Petersen, and V. Krüger, "A Comparison of Iterative 2D-3D Pose Estimation Methods for Real-Time Applications," in *Scandinavian Conference on Image Analysis*, 2009, pp. 706-715.

[41] V. Lepetit, F. Moreno-Noguer, and P. Fua, "EPnP: An Accurate O(n) Solution to the PnP Problem," *International Journal of Computer Vision,* vol. 81, pp. 155-166, 2009.

[42] D. Wagner and D. Schmalstieg, "ARToolKit on the PocketPC Platform," in *International Augmented Reality Toolkit Workshop*, 2003, pp. 14-15.

[43] M. Mohring, C. Lessig, and O. Bimber, "Video See-Through AR on Consumer Cell-Phones," in *International Symposium on Mixed and Augmented Reality*, 2004, pp. 252-253.

[44] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "MonoSLAM: Real-Time Single Camera SLAM," *Transactions on Pattern Analysis and Machine Intelligence,* vol. 29, pp. 1052-1067, 2007.

[45] L. Taehee and T. Hollerer, "Hybrid Feature Tracking and User Interaction for Markerless Augmented Reality," in *IEEE Virtual Reality Conference*, 2008, pp. 145-152.

[46] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg, "Real-Time Detection and Tracking for Augmented Reality on Mobile Phones," *IEEE Transactions on Visualization and Computer Graphics,* vol. 16, pp. 355-368, 2010.

[47] H. Hile and G. Borriello, "Information Overlay for Camera Phones in Indoor Environments," in *International Symposium on Location and Context Awareness*, 2007, pp. 68-84.

[48] T. P. Morgan. (2012). *ARM Snags 95 Percent Of Smartphone Market, Eyes New Areas For Growth*. Available: http://www.crn.com/news/components-peripherals/240003811/arm-snags-95-percent-of-smartphone-market-eyes-new-areas-for-growth.htm

[49] W. Garage. *Open Source Computer Vision Library*. Available: http://opencv.willowgarage.com/wiki/

[50] FourCC. *YUV to RGB Conversion*. Available: http://www.fourcc.org/fccyvrgb.php

[51] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua, "BRIEF: Computing a Local Binary Descriptor Very Fast," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 34, pp. 1281-1298, 2012.

[52] A. Sawchuk, "Girl (Lena, or Lenna)," ed, 1972.

[53] A. K. J. a. R. C. Dubes, *Algorithms for Clustering Data*: Prentice-Hall, 1981.

[54] M. Weinmann, S. Hinz, and B. Jutzi, "Fast and Automatic Image-Based Registration of TLS Data," *ISPRS Journal of Photogrammetry and Remote Sensing,* 2011.

[55] M. Fischler and R. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the ACM,* vol. 24, pp. 381-395, 1981.

[56] A. Z. Richard Hartley, *Multiple View Geometry in Computer Vision*, Second ed.: Cambridge University Press, 2004.

[57] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME - Journal of Basic Engineering,* vol. 82, pp. 35-45, 1960.

[58] J. J. LaViola, "Double Exponential Smoothing: An Alternative to Kalman Filter-Based Predictive Tracking," in *Workshop on Virtual Environments*, Zurich, Switzerland, 2003, pp. 199-206.

[59] W. Garage. (2012). *OpenCV Change Log*. Available: http://code.opencv.org/projects/opencv/wiki/ChangeLog

[60] A. Ufkes and M. Fiala, "A Markerless Augmented Reality System for Mobile Devices," in *Canadian Conference on Computer and Robot Vision*, Regina, Saskatchewan, 2013.

[61] J. Hruska. (2013). *Nvidia's Tegra 4 Demystified: 28nm, 72-core GPU, Integrated LTE, and Questionable Power Consumption*. Available: http://www.extremetech.com/computing/144942-nvidias-tegra-4-demystified-28nm-72-core-gpu-integrated-lte-and-questionable-power-consumption

[62] K. Group. (2008). *OpenCL - The Open Standard for Parallel Programming of Heterogeneous Systems*. Available: http://www.khronos.org/opencl/

[63] AccelerEyes. (2011). *Mobile GPU Computing for Accelerated Apps*. Available: http://www.accelereyes.com/products/mobile