QUALITATIVE IMAGE BASED
LOCALIZATION IN A LARGE
BUILDING

by
Christopher Tyler Card
A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science (Computer Science).

Golden, Colorado
Date ______________________

Signed: ______________________
Christopher Tyler Card

Signed: ______________________
Dr. William Hoff
Thesis Advisor

Golden, Colorado
Date ______________________

Signed: ______________________
Dr. Randy Haupt
Professor and Head
Department of Electrical Engineering and Computer Science
ABSTRACT

Interest in indoor localization is growing because it is an important component of many applications (e.g. augmented reality, customer navigation). Image-based localization, using naturally-occurring features in the environment, is an attractive solution to this problem. A challenge is to be able to perform this on a mobile device with limited computing power. Another challenge is that buildings can have interior locations with similar appearances, which can confuse an image-based recognition system. Since many applications do not need the exact location of an image, this research focuses on qualitative localization, which is the problem of determining the approximate location by matching a query image to a database of images. This paper proposes a novel approach that uses an efficient hashing scheme to quickly identify candidate locations, then applies a strong geometric constraint to reject matches that have similar appearance. Through experiments using a large campus building, the approach is shown to be able to localize a query image with high accuracy and have the potential to run in real time on a mobile device.
TABLE OF CONTENTS

ABSTRACT ................................................................. iii

LIST OF FIGURES ......................................................... vi

LIST OF TABLES ........................................................... vii

ACKNOWLEDGMENTS ...................................................... viii

CHAPTER 1 INTRODUCTION ............................................. 1

CHAPTER 2 RELATED WORK ............................................. 3
  2.1 Detecting Image Features ......................................... 3
  2.2 Feature Matching .................................................. 6
  2.3 Image Matching .................................................... 8

CHAPTER 3 LOCALIZATION ALGORITHM ............................ 14
  3.1 Feature Detection .................................................. 14
  3.2 Feature Matching .................................................. 14
  3.3 Verification ........................................................ 15
  3.4 Local Map .......................................................... 17

CHAPTER 4 EXPERIMENTS ............................................. 18
  4.1 Database ............................................................ 18
  4.2 Tests ................................................................. 20
    4.2.1 Parameter Evaluation ...................................... 20
    4.2.2 Subsample Test .............................................. 22
    4.2.3 Local Map Test ............................................. 23
CHAPTER 5 CONCLUSION AND FUTURE WORK .......................... 27

5.1 Future Work ......................................................... 27

5.2 Conclusion ......................................................... 28

REFERENCES CITED .................................................. 30

APPENDIX - SUPPLEMENTAL ELECTRONIC FILES ....................... 32
LIST OF FIGURES

Figure 2.1  SIFT descriptor ......................................................... 4
Figure 2.2  Example of ORB features ............................................. 5
Figure 2.3  FAST corner detection .................................................. 6
Figure 2.4  Epipolar geometry ....................................................... 10
Figure 2.5  Example database images .............................................. 12
Figure 2.6  More example database images ....................................... 13
Figure 3.1  Illustration of spacial consistency voting ......................... 16
Figure 4.1  Shows the image capture location .................................... 19
Figure 4.2  ROC graphs ............................................................... 21
Figure 4.3  Example of image changes causing incorrect matches ........ 24
Figure 4.4  Four examples of true positive ....................................... 25
Figure 4.5  Four examples of false positives ..................................... 26
Figure 5.1  An example of an epiline .............................................. 28
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Subsample test results</td>
<td>23</td>
</tr>
<tr>
<td>4.2</td>
<td>Local map test results</td>
<td>25</td>
</tr>
<tr>
<td>A-1</td>
<td>Supplemental electronic files</td>
<td>32</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENTS

I would like to thank Dr. William Hoff for his help guiding this thesis work and providing invaluable insight.
CHAPTER 1
INTRODUCTION

The goal of indoor localization is to determine the location of a mobile device in an indoor environment. Interest in this problem is growing because determining location inside a building is an important component of many applications, such as augmented reality, customer navigation, and behavior and movement tracking. Once the location of a device is found many useful tasks can be done, e.g. providing navigation directions or providing location-specific information to the user.

The problem is very different from outdoor localization because the device no longer has access to a reliable GPS signal. A variety of alternative methods can be used in place of GPS. The most popular approaches require some kind of infrastructure to be present in the building. In the most recent Microsoft Indoor Localization Competition at IPSN 2015, 30 of the 48 submissions required infrastructure. For example, Tesoriero et al [1] places Radio Frequency IDentification (RFID) markers throughout the environment. This has the drawback that the number of markers increases with the size of the building, which increases the cost. Another problem is that if any markers fail or are moved the system also fails, unless the placement of RFID markers is quite dense. Another approach is to use Wi-Fi fingerprinting as done by Hile et al [2]. Wi-Fi based systems also experience some of the same problems as RFID. For instance, if the Wi-Fi routers go offline or are changed the entire system needs to be re-calibrated. Another problem that affects Wi-Fi is that multi-story structures cause signal reflection and interference which degrades the system’s performance.

An alternative to RFID and Wi-Fi is to use image-based localization. This is attractive because the vast majority of people already have mobile devices (e.g. smart phones) with cameras and the approach is applicable to buildings without RFID or Wi-Fi. A recent survey of optical indoor positioning systems is given in [3]. One approach to image-based lo-
calization is to place specially designed fiducial markers in the environment, at pre-measured locations (e.g. [4]). The fiducial markers are easy to recognize in images and are cheap to make. However, this approach still has many of the drawbacks of signal-based indoor localization, i.e. the need to install additional infrastructure and the likelihood of system failure if the markers are removed or damaged.

Instead of using fiducial markers, image-based localization can use naturally-occurring features in the environment. This has the advantage that no infrastructure is required. One challenge is that doing localization based on naturally-occurring features can be computationally intensive, but it is desirable to have the application run on a mobile device with limited computing power. Another challenge is that in a large building there can be many locations that have a similar appearance, thus potentially confusing an image-based recognition system.

This paper presents a novel approach that can perform localization within a large building using no infrastructure or any special mapping steps. The above challenges are addressed in the following ways: first, an efficient hashing scheme is used to quickly identify candidate locations that match a query image. Next, a strong model based on geometric constraints is employed to identify the correct match, while rejecting matches that have a similar appearance. Finally, a local map in the vicinity of the user is constructed to limit the search for candidate matches. Although the system was not implemented on a mobile device, an initial analysis shows that it has the potential to run in real time on a reasonably capable mobile device.

The remainder of this paper is organized as follows. Chapter 2 describes previous related work that motivates this approach. Chapter 3 describes the approach in detail. Chapter 4 provides an evaluation of the system on a large campus building. Chapter 5 is the conclusion.
There is a wide spectrum of work regarding indoor localization using signals and computer vision. This research focused on computer vision without the use of additional infrastructure, such as fiduciary markers or signals, to perform localization. This chapter describes the related work in indoor localization using computer vision. There are three broad categories that were necessary to consider: detecting image features, matching feature points, and matching to an image.

2.1 Detecting Image Features

Image features are key points that represent an area of interest within an image and can be consistently found.

The most widely used and state-of-the-art natural image feature descriptor is Scale Invariant Feature Transform (SIFT) [5]. SIFT is a feature descriptor that is robust to image noise, intensity variations and some affine deformations [5]. The SIFT descriptor defines a 16x16 region around a feature point that can be consistently computed in images and matched reliably to other images. The 16x16 regions’ gradients are represented by a 4x4 histogram with 8 orientations in each cell (see Figure 2.1).

There has been work to extend the usefulness of SIFT by reducing the size of the 128-dimensional feature vector while maintaining the robustness of SIFT. One way Yan et al [6] worked to reduce the dimensionality of the SIFT feature vector is by applying a Principal Component Analysis (PCA) to reduce the 128-dimensional feature vector down to a 36-dimensional feature vector. Yan et al calls this method PCA-SIFT and claims that it not only reduces the SIFT feature vector dimensionality but also improves matching speed [6]. One researcher that applies PCA-SIFT to indoor localization is Hisato et al [7]. Hisato et al’s goal is to determine a user’s location within a train museum so the user can receive
information about exhibits based on pictures taken from the user’s mobile phone. A major drawback of SIFT is that SIFT is expensive to compute because SIFT calculates all the features in scale space [5]. This process requires a lot of memory and processor time making it difficult to do on a mobile device, especially in real time.

Another state-of-the-art natural image feature descriptor is Speeded Up Robust Feature (SURF) [8]. SURF uses a different detection and descriptor approach than SIFT. The major differences between SIFT and SURF is that SURF uses integral images to approximate scale space and Haar wavelets [8]. One of the benefits that SURF has over SIFT is that it is faster to compute while maintaining robustness that is equivalent to SIFT [8]. Even though SURF is faster, it is still too computationally expensive to be computed on a mobile device in real time.

A more recently developed natural image feature descriptor that attempts to address the computational complexity of SIFT and SURF and is designed for mobile devices is Oriented FAST Rotated BRIEF (ORB) [9]. ORB was designed by Rublee et al [9] to be a faster and less computationally expensive alternative to SIFT and SURF while maintaining a similar robustness with respect to image deformations (see Figure 2.2). ORB does not introduce any new algorithms to calculate the features. However, ORB does modify and combine two
existing algorithms. The algorithm that ORB uses to identify feature points is Features from Accelerated Segment Test (FAST). FAST finds corners by examining pixels along an arc around a point $p$. If the arc passes through more than $n$ continuous pixels that have an intensity above a threshold $t$, then $p$ is determined to be a corner (see Figure 2.3) [10].

To make FAST more robust, Rublee et al modified it in two ways. The first way FAST was modified was by taking more than $N$ of the key points that FAST identifies, applying a Harris corner measure to sort them, and then taking the top $N$ of the sorted key points [9]. The second modification Rublee et al made after finding the top $N$ key points was aimed at finding their orientation. To do this they used Intensity Centroids (IC), because ICs were found to be more robust than using gradient-based methods to determine orientation [9]. To get the orientation of the key point, ORB finds the vector from the key point to the IC of the area around the key point, creating Oriented FAST.

To describe the key points, Rublee et al used the Binary Robust Independent Elementary Features (BRIEF) descriptor. BRIEF is formed from tests $\tau_{1-n}$ on pixel intensity in image patches $p_{1-n}$, each of which are of size $SxS$. The tests are then used to form $n_d$-dimensional
bitstrings \( f_{n_d}(p_{1-n}) \) [11]. Rublee et al modified BRIEF in two ways. They initially took the orientations calculated for the feature points and used them to create “steered” BRIEF. However, this was not a sufficient modification to achieve the desired accuracy because BRIEF and steered BRIEF have a high correlation and therefore, are not very discriminative [9]. To solve this issue Rublee et al took the tests created while forming steered BRIEF and applied a greedy algorithm to the tests. This greedy algorithm finds uncorrelated tests that have a mean near 0.5 and repeats itself until 256 of the tests are used [9]. This makes steered BRIEF more discriminative and, therefore, more robust which results in ORB [9]. Rublee et al’s experiments demonstrate these claims which have been confirmed by Heinly et al [12]. Rublee et al also demonstrated that ORB is capable of being computed on a video stream of 7Hz on a 1GHz ARM processor with 512Mb of RAM, which represents a mid to high-end mobile phone [9]. This result makes it a promising descriptor to use for a real time application on a mobile device.

### 2.2 Feature Matching

Feature matching is important because it gives a good approximation of the similarity between images. Most of these techniques are used in conjunction with natural image feature descriptors. One of the most common feature matching techniques is the Brute Force (BF)
matching method. This method compares a descriptor from one image to all the descriptors of the other images to find the closest match in feature space. This is by far the most accurate method of feature matching, as it is exhaustive, but still has drawbacks. The most prevalent of these drawbacks is that it is only practical on small sets of images (<500); once the database of images becomes large, the time to match features becomes prohibitively expensive.

To avoid this problem, the database descriptors are stored in a hash table. The same hash is applied to a query descriptor. The database descriptors at that location in the hash table are then retrieved. This is a very efficient and fast operation, especially for large databases. Locality Sensitive Hashing (LSH) [14] was used which is a hashing technique that preserves the locality\(^1\) of key points when generating the hash of the image descriptor. In other words, the difference between hash values is a good approximation of the distance between the points in feature space. This allows finding nearby descriptors in feature space, not just the descriptors at the hash location. This is important because image deformation and noise can cause the descriptors to change. The algorithm finds the \(k\) nearest neighbors for the query point, where \(k = 15\).

The main idea behind LSH is that if two features are close to each other in a high-dimensional space they will remain close together after a projection into a low-dimensional space. LSH forms the hash of the query vector \(\vec{q}\) by first performing a scalar projection (i.e. dot product) of \(\vec{q}\), given by \(h(\vec{q}) = \vec{q} \cdot \vec{x}\) where the values of \(\vec{x}\) are drawn randomly from a \(p\)-stable distribution. The scalar projection \(h(\vec{q})\) is then quantized into hash bins with the intention that points that are close in feature space should end up in the same hash bin. Conversely points that are distant in feature space should end up in different hash bins. To increase the chances of this happening, \(k\) dot products of \(\vec{q}\) are performed in parallel with \(k\) random independent vectors \(\vec{x}\). This results in a \(k\)-dimensional index \(I\). To get the final index (i.e. hash value) the dot product of \(I\) and a vector of integer weights \(H\) is taken.

\(^1\)The relative location of a point in regards to other points in the image.
A stable distribution is known as a $p$-stable distribution if the following is true: if there is a $p \geq 0$ such that for any $n$ real numbers $v_i, i = 1, \ldots, n$ and i.d.d.\footnote[2]{independent and identically distributed} variables $X_i, i = 1, \ldots, n$ with distribution $\mathcal{D}$ then $\sum_i v_i X_i \overset{d.}{=} \left( \sum_i |v_i|^p \right)^{1/p} X$ where $\overset{d.}{=} \text{represents distribution equality}$ and $X$ is a random variable with distribution $\mathcal{D}$ [15]. This means that a stable distribution is $p$-stable if $n$ different random independent variables with distribution $\mathcal{D}$ can be summed to have the same distribution as a random variable, with distribution $\mathcal{D}$, multiplied by a scalar value. For instance, the Gaussian (normal) distribution $\mathcal{D}_G$ defined by the density function $g(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$, is a 2-stable distribution [14].

One of the approaches that used this technique is Hisato et al’s [7]. They use a version of LSH called E-LSH which measures the distance between the randomly chosen vector from the $p$-stable distribution and the key point in Eigen space. Hisato et al found that this is an effective approach for matching the features of a query image to a large database of image features. However, they experienced one of the problems that plagues feature matching: the occurrence of incorrect feature matches. This can also occur with BF matching.

These incorrect matches can be caused by the feature matching algorithms choosing between two points that are similar to the query point, but only one is correct. This is the result of image deformations (e.g. image blurring, out of plane rotations) or locations with similar appearances. Either of these problems alone makes feature matching methods an unreliable, but necessary, step to perform image matching.

\subsection*{2.3 Image Matching}

Once the features are matched to one or more images in the database, a further step is needed to match an image to another image to correct the feature matching problem. The most common and effective way to do this is to fit the matched key points to a model and find the number of inliers to that model. The most commonly used models are homographies and fundamental matrices[16] but others use their own custom models. For instance, Hile \etal [2] used the building’s floor plan as a model to match their query image to. However,
custom models can be difficult to develop and can introduce new errors to the system.

The homography model tries to fit a homographic matrix, \textit{i.e.} a matrix defining the transformation of one plane to another, to a set of matched features. In contrast, fitting the points to the fundamental matrix model attempts to find a matrix that defines the epipolar geometry between two images to some scale factor (see Figure 2.4) \cite{16}. The fundamental matrix is derived as follows:

\[
\overrightarrow{C_0p_0} \cdot \left( \overrightarrow{C_0C_1} \times \overrightarrow{C_1p_1} \right) = 0 \tag{2.1}
\]

where \(\overrightarrow{C_0p_0}, \overrightarrow{C_0C_1}, \text{ and } \overrightarrow{C_1p_1}\) are co-planar. Treating \(p_0\) and \(p_1\) as normalized image points, Equation (2.1) can be written as

\[
p_0 (t \times Rp_1) = 0 \tag{2.2}
\]

where \(t = C_0t_{C_1\text{org}}\) is the translation vector of Camera 1 origin with respect to Camera 0; \(R = C_0C_1R\) is the rotation of Camera 1 with respect to Camera 0. However, the cross product of \(t\) and \(R\) is equivalent to the 3x3 skew symmetric matrix of \(t\) multiplied by \(R\). Therefore, Equation (2.2) can be written as

\[
p_0^T [t]_x Rp_1 = 0 \tag{2.3}
\]

where \([t]_x\) is the 3x3 skew symmetric matrix of \(t\). Letting \(E = [t]_x R\), Equation (2.3) can be written as

\[
p_0^T Ep_1 = 0 \tag{2.4}
\]

where \(E\) is the essential matrix. If \(u_0\) and \(u_1\) are the corresponding unnormalized points to \(p_0, p_1\) then

\[
p_0^T = \left( K^{-1} u_0 \right)^T = u_0^T K^{-T} \\
p_1 = K^{-1} u_1 \tag{2.5}
\]
where $K$ is the intrinsic camera matrix and then substituting Equation (2.5) into Equation (2.4) yields

$$u_0^T K^{-T} EK^{-1} u_1 = 0. \quad (2.6)$$

Identifying the fundamental matrix as $F = K^{-T} EK^{-1}$, Equation (2.6) then becomes

$$u_0^T Fu_1 = 0. \quad (2.7)$$

Since Equation (2.7) is a system of homogeneous equations, $F$ is only known to some scale factor. This means that any scalar multiple of $F$ is also a solution to Equation (2.7).

To build the fundamental matrix (i.e. find $F$ that satisfies Equation (2.7)) there are many different algorithms but the most commonly used and effective algorithm is RANdom SAmple Consensus (RANSAC) [17]. RANSAC attempts to fit a set $S$ of matched points to a model $M$. The first step is to find a subset $S^*$ of matched points from points $p \in S$ that fit $M$ within an error tolerance $\epsilon$. If $\|S^*\| \geq t$, the model $M$ is returned; otherwise, $t$ is adjusted and the process is repeated until a failure point is reached [17]. This process maximizes the number of matched features that are inliers to the model. RANSAC is also used by Hile et al [2] to match their floor plane (comprised of line segments) to their model which is the floor plan of the building.
Another method for image matching is the Bag of Words (BoW) approach. BoW quantizes feature vectors into visual words, thus creating a visual vocabulary [18]. To match a query image to an image in a database, the algorithm simply finds the distribution (histogram) of visual words found in the query image and compares this distribution to those found in the database images. Although this could be used for qualitative localization, BoW can fail when the histograms of visual words are too similar.

In a large building, there can be many locations that have a similar appearance. Walls and floors often have little or no texture and doors look very similar. For example, Figure 2.5 and Figure 2.6 shows sets of images from a large building on the Colorado School of Mines (CSM) campus. There are many features (such as the corners between doors and the floor) which are present in all the images. Thus, the histograms of words would not be very distinctive. This would result in incorrect matches to the database.
Figure 2.5: Examples database images depicting scenes that have similar features but are captured at different locations.
Figure 2.6: More example database images.
The localization algorithm is logically divided into three steps which are discussed in the subsections below: (1) feature detection, (2) feature matching, and (3) verification. This chapter is concluded with a description of the “local map” method.

Initially, the original camera images were reduced to 600x800 pixels. The reduced images have the same aspect ratio of the original images to avoid introducing distortions. The smaller images allow faster computation with no significant loss of accuracy.

3.1 Feature Detection

ORB [9] was chosen as the feature detection algorithm since it provides robustness to image deformation that is close to SIFT and SURF while providing an increase in computational speed of an order of magnitude [9]. This makes it ideal for localization in real time on a mobile device. In the experiments, ORB descriptors were computed for a query image in 0.05 seconds. The ORB descriptors for the database images were precomputed.

3.2 Feature Matching

Given a set of ORB descriptors extracted from a query image, these descriptors then need to be matched to the descriptors from the database of images. LSH is used to do this because it is extremely fast when matching features against a large database (1,073,903 feature points). In the experiments, LSH was able to match against a large database in about 1.47 seconds and it was able to match against a local map (containing 33,000 feature points) in about 0.095 seconds. However, the speed of LSH is comparable to that of BF matching for small databases (i.e. the local map on the mobile device). This makes BF matching a better choice when the database is small because of its higher accuracy.
The potential matches for each query point, \( q_i \), are then filtered using two steps, as described below:

1. **Ratio test.** The first step of the filter process is to determine if there are multiple feature points from one database image that are nearest neighbors to \( q_i \). If this is the case, the closest feature point from the database image has to be 80% closer to \( q_i \) than the second closest feature point. If it is not, then all feature points from that database image that are nearest neighbors to \( q_i \) are ignored. If the closest feature point is 80% closer than the next closest feature point, then the closest feature point is kept and the rest are discarded. This is done because two feature points that are both roughly the same distance to \( q_i \) are not likely be very descriptive features.

2. **Spatial consistency test.** The next step checks whether each matched pair of points is spatially consistent. The approach of Sivic et al [19] is used for this step (see Figure 3.1). The idea is that neighbors of the query point should have matches that are neighbors of the database point. Here, “neighbors” means that the points are neighbors in image space, not feature space. If the number of spatially consistent neighbors is below a threshold (a threshold of 6 points was used in this work), then the potential match is discarded.

After the two filtering steps are completed, the two database images with the highest number of matches to the query image are selected. These are the candidate matches to the query image. If there is a tie for second place, all images that are tied are kept.

### 3.3 Verification

The verification step of the algorithm tests each candidate database image to see if the matching points fit a geometric constraint with the query image. The model used for the geometric relationship is the fundamental matrix [16]. RANSAC [17] is used to eliminate outliers. A fundamental matrix is found between the query image and every candidate database image. Then the image with the most inliers is found. If the number of inliers
exceeds a threshold (described in Chapter 4), then that database image is determined to be the correct match. If not, then all the candidate images are passed to a secondary processing step.

The secondary processing step rematches all image features in the query image to the candidate set of images, except that it now uses BF instead of LSH to match image features. The resulting image matches are filtered and a fundamental matrix is again fitted between each candidate database image and the query image. The image with the most inliers is found and, if the number exceeds the threshold, it is determined to be the correct image; otherwise, the query image is considered to have no acceptable match in the database. This secondary processing step is done because LSH is not as accurate as BF matching but the correct image may still be in the candidate set of images. Therefore, BF matching is used because it is more likely to find a larger number of correct feature matches between the query image and the candidate images. This increases the likelihood that the correct match will be found if the correct match is in the candidate set of images.
3.4 Local Map

If the size of the database can be reduced this would potentially speed up computation as well as improve the accuracy of matching. To do this, this research proposed using a “local map” in the vicinity of the user which contains only the database images near the current location of the user. The size of the local map depends on how fast a user can reasonably walk in a given amount of time. So long as the user is within the boundaries of the local map, localization queries can be done by matching to that local map.

The concept is as follows: When a user first runs the system to perform localization, the image is sent to a server which matches the query image to the entire image database. Once the user’s approximate location is found, the system sends a local map to the user’s mobile device. The user’s mobile device then utilizes the this map to perform localization. This greatly speeds up processing and improves accuracy which allows the mobile device to perform localization in real time. When the user approaches the edge of the current local map, the server sends the next local map to their mobile device. Although this concept was not implemented, the potential benefits of using a series local maps in terms of run time and accuracy were evaluated, as described in the next chapter.
CHAPTER 4
EXPERIMENTS

This chapter describes the database used and details the methodology applied to test the algorithm. The algorithm was implemented using C++ and the open source software, OpenCV. The algorithm was tested on a laptop running Windows 7 with a 2.6 GHz processor and 4GB of RAM. For the local map test (described below) this laptop was used to approximate an actual mobile device.

4.1 Database

The database was captured using a Cannon Rebel t2i Single Lens Reflex (SLR) camera with 8 megapixels per image. The database was captured in Brown Hall at CSM which is a large (100,000 square foot) building containing offices, classrooms and laboratories. The database consists of the 1st, 2nd, and 3rd floors of Brown Hall, as these floors have a representative sample of indoor environments which contain sparse texture and similar structural features. For each of the specified floors, the images were taken every 5 feet in a zigzag fashion, on the assumption that people do not typically walk down halls in a perfectly straight line (see Figure 4.1). At each position multiple images were taken facing both directions in the hall and additional directions to capture the appearance of nearby characteristic features (e.g. doors, side halls). The location where each image was taken was physically measured and recorded. The operation of the system is not dependent upon knowing these locations. The measurement of image locations were recorded solely to test the system’s accuracy.

The database is comprised of 1,382 images with a total of 1,073,903 feature points. Images in the database overlap, meaning that adjacent images typically contain a portion of the same scene (see Figure 2.5 and Figure 2.6 for example images).
Figure 4.1: Shows the image capture locations in Brown Hall, indicated by the red dots. (a) Is the 1\textsuperscript{st} floor, (b) is the 2\textsuperscript{nd} floor and (c) is the 3\textsuperscript{rd} floor.
4.2 Tests

The following subsections describe the tests performed to evaluate the algorithm using the collected database. A match is deemed to be correct if the location of the database image is less than 21 feet from the query image. In the tests, the correct match to a query image was in the database about 92% of the time. The correct match may not have been in the database because the environment changed or the correct match was removed as part of the test set.

4.2.1 Parameter Evaluation

One of the most important parameters in the algorithm is the threshold for the number of inliers to a fundamental matrix, which determines if a query image is successfully matched. The theoretical minimum number of point correspondences necessary to compute a fundamental matrix is five, although usually the correct match to a query image has well above five inliers. If the number of inliers is low, this could mean that the query image has no correct match in the database and is matching an incorrect image by accident.

To avoid incorrect matches, it is desirable to use a higher threshold for the required number of inliers. This reduces the probability of a false match. However, this also reduces the probability of a true match. Conversely, lowering this threshold makes it more likely that a query image will be successfully matched to the correct database image. However, if a query image actually has no correct match in the database, lowering the threshold also increases the probability that a false match will occur.

To evaluate the effect of changing (i.e. tuning) this parameter on the probability of getting a false match, the following study was done. Thirty-five (35) images were randomly chosen and removed from the database. Each of these 35 images had a correct match in the database. Thirty-five other images were captured (using the same camera) from parts of the building that were not in the database. These images had no correct match in the database.
Figure 4.2: ROC graphs, where (a) is a general example of a ROC curve and (b) is the ROC curve for a test set of 70 images matched against the remainder of the database. The same test set was used for all 8 runs.

A Receiver Operating Characteristics (ROC) curve was generated (see Figure 4.2(a)). ROC curves are based on a 2x2 confusion matrix, which records the count of the four possible outcomes of running the localization algorithm at each setting of the algorithm parameter. The four possible outcomes are:

- **True Positive (TP)**- The correct match to the query image was in the database and the system found the correct match.

- **True Negative (TN)**- The correct match to the query image was not in the database and the system correctly decided that there was no match.

- **False Positive (FP)**- The correct match to the query image was not in the database, but the system matched it to an incorrect image.

- **False Negative (FN)**- The correct match to the query image was in the database, but the system was unable to find a match.

The True Positive Rate (TPR) is defined as the ratio of true positives (TP) to the total number of positives (TP+FN). The False Positive Rate (FPR) is defined as the ratio of false positives to the number of total negatives (FP+TN) [20].
The ROC curve is formed by plotting TPR against FPR. The resulting ROC curve is shown in Figure 4.2(b). As can be seen, the TPR is fairly high for most parameter settings. For example, using a threshold of 16, the TPR is about 94%, meaning that if the correct match is in the database the system will find it 94% of the time. The FPR for this case is about 17%, meaning that when the correct match is not in the database, the system finds an incorrect match instead of outputting a “no match” decision. Although this FPR seems high, the number of cases where there is no correct match in the database is small, so this outcome is relatively rare.

4.2.2 Subsample Test

To assess the overall accuracy of the algorithm over multiple runs, a subsample test was performed. Twenty test sets were created, where each test set consisted of 30 randomly chosen images from the full database, with no restriction on proximity. For each test set, the 30 images were removed from the database and then used to query the database. In this experiment a threshold of 16 inliers was used as the decision threshold; this means that a matching image is deemed to be valid if it contains at least 16 inliers to the fundamental matrix.

Overall, the algorithm performed well. Combining the results from all 20 subsample tests, the algorithm achieved an accuracy of 92.5% (see Table 4.1). Here accuracy is defined as the fraction of all outcomes that were correct. Specifically, it is the number of outcomes in rows 1 and 5 of Table 4.1 divided by the total number of trials, row 6. These results show that the algorithm can localize a query image with a high degree of confidence. An FN example is shown in Figure 4.3(a)(b), where (a) should have matched (b) but did not pass the decision threshold because an insufficient number of inliers were found. Four examples of TPs are shown in Figure 4.4. Four examples FPs in which the query image had a match in the database but the algorithm found an incorrect match are shown in Figure 4.5. The FPs were caused by a set of highly clustered points.
Table 4.1: Subsample test results. Each part of the table contains the sum from 20 subsample tests.

<table>
<thead>
<tr>
<th>Outcome</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query has a match in database and algorithm found a correct match</td>
<td>538</td>
</tr>
<tr>
<td>Query has a match in database and algorithm found an incorrect match</td>
<td>14</td>
</tr>
<tr>
<td>Query has a match in database and algorithm declared “no match”</td>
<td>27</td>
</tr>
<tr>
<td>Query has no match in database and algorithm found an incorrect match</td>
<td>4</td>
</tr>
<tr>
<td>Query has no match in database and algorithm declared “no match”</td>
<td>17</td>
</tr>
<tr>
<td>Total number of queries</td>
<td>600</td>
</tr>
</tbody>
</table>

Another important factor for this algorithm is the amount of time it takes to match a query image to the database. On the laptop the average time to match a single query image to the full database (minus the 30 images for each test set) was 6.22 seconds. While not especially fast, this only needs to be done once, when the user first performs the localization step. After that the localization steps are performed with a local map that has a much smaller database of images. These steps are much faster, as is described in the next subsection.

4.2.3 Local Map Test

Once the first query image is localized by the server, the mobile device receives a local map of images surrounding its current location. The number of images in the local map is chosen so that users will likely remain within this local map for only a short time. Thus, all queries performed in that time frame will most likely correctly match to an image in the local map. The motivation for using a local map is that it will reduce the time to perform a query as well as improve the accuracy of the algorithm.

In this test, 65 images were used to form the local map because that number of images approximates the distance a user who is unfamiliar with a building would travel in about 20 seconds. A small local map reduces the amount of data that needs to be transferred to the mobile device and the frequency with which the device needs to contact the server.
Figure 4.3: Examples of image changes causing incorrect matches. (a) Should have matched (b) but was reported as a negative match because the image illumination changed, causing a FN. (c) Incorrectly matched the database image (d), this FP is the result of a change in the environment that caused (c) to have a similar appearance to (d).

Since the system has not yet been tested on a mobile device, the exact size of the local map was approximated in this test. Therefore, the approximate time to localize a query image is expected to vary with the size of the local map.

For query images, 19 test images from the 3\textsuperscript{rd} floor of Brown Hall were captured independently from the database using the same camera. The local map consisted of 65 images from the collected database that contained the area around the test images.

The results from localization using the local map yielded an accuracy of 94.74\% (see Table 4.2). The fact that the accuracy of the test is not closer to 100\% is because of minor changes to the environment between the time that the database images and the query images were captured (see Figure 4.3(c)(d)). However, changes like this are to be expected as the indoor environment is not static. If the environment changes the database needs to be updated. This is a problem for this approach as well as the other approaches researched ([2], [7], [21]).

The system took an average of 1.902 seconds to localize a query image. These results show that the algorithm has the potential to run on a mobile device in real time.
Table 4.2: Local map test results.

<table>
<thead>
<tr>
<th>Outcome</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query has a match in database and algorithm found a correct match</td>
<td>16</td>
</tr>
<tr>
<td>Query has a match in database and algorithm found an incorrect match</td>
<td>1</td>
</tr>
<tr>
<td>Query has a match in database and algorithm declared “no match”</td>
<td>0</td>
</tr>
<tr>
<td>Query has no match in database and algorithm found an incorrect match</td>
<td>0</td>
</tr>
<tr>
<td>Query has no match in database and algorithm declared “no match”</td>
<td>2</td>
</tr>
<tr>
<td>Total number of queries</td>
<td>19</td>
</tr>
</tbody>
</table>

Figure 4.4: Four examples of TP matches; (a),(c),(e), and (g) are the retrieved database images to their respective query images (b),(d),(f), and (h). The black lines are the epipolar lines found using the fundamental matrix and the pink numbers are points that are inliers to the fundamental matrix.
Figure 4.5: Four examples of FP matches where the query image had a match in the database but the algorithm found an incorrect match; (a),(c),(e), and (g) are the retrieved database images to their respective query images (b),(d),(f), and (h). The black lines are the epipolar lines found using the fundamental matrix and the pink numbers are points that are inliers to the fundamental matrix.
CHAPTER 5
CONCLUSION AND FUTURE WORK

This chapter discusses some of the future work that could be undertaken to improve the algorithm and then summarizes what has been accomplished over the course of this research.

5.1 Future Work

Though this research has accomplished what it set out to do, there is always room for improvement and expansion. Potential future work could be to continue to improve the algorithm’s speed and accuracy and/or to adapt the algorithm to a mobile device.

One way to improve the accuracy of the algorithm is to increase the number of feature point matches between the retrieved database image and the query image and then recalculate the fundamental matrix. One way to increase the number of feature matches between the retrieved database image and the query image is to use epilines. Since the fundamental matrix is already calculated between the two images, it can be used to determine the epiline in the database image that corresponds to a point in the query image. Then, for each feature point in the query image that does not have a match in the database image, an epiline is calculated. For each epiline feature points in the database image that intersect it are found. Of the feature points that intersect the epiline, the feature point that best matches the query feature point that created the epiline is found. This is done because the feature point in the database image that best matches the query feature point will most likely lie on the query feature point’s corresponding epiline. For example, Figure 5.1 shows a red epiline in the left image that intersects a point that is a good match to the epiline’s corresponding point in the right image.

There are many different mobile devices that the algorithm could be adapted to. The mobile device that the algorithm would most likely be adapted to is a high-end Android phone (e.g. HTC Desire EYE, Samsung Galaxy Note 4). This is because a large number
Figure 5.1: An example of an epiline. The right image is the database image and the left is the query image. The red line in the database image is the epiline that corresponds to the point in the query image.

of people own them and it is free to develop on the Android platform. Finally using an Android phone relates to the use case that inspired this research: to guide a user through an indoor environment using a mobile phone and natural-occurring features in the environment. Another mobile device that the algorithm could be adapted to is the NVIDIA Jetson TK1 because it runs Linux and is used in robotics and other applications. Regardless of which mobile device the algorithm is adapted to, the server program would need to be written and the communication protocol would need to be optimized.

5.2 Conclusion

This paper has presented a novel approach to indoor localization that does not require any additional infrastructure or any special mapping techniques. Using only naturally-occurring features in the environment it was demonstrated that this approach can qualitatively localize an image in a large building with a high degree of confidence. The results also show that the use of a local map around the mobile device’s known location improves the accuracy of localization. Although the approach was not implemented on a mobile device, analysis
shows that it has the potential to run in real time on such a device.
REFERENCES CITED


APPENDIX - SUPPLEMENTAL ELECTRONIC FILES

The included electronic files (See Table A-1) are the source code used to run and test the core algorithm, the code used to form the basic map and find the connected components of the generated graph.

Table A-1: This table contains the supplemental electronic files that are used with this thesis.

<table>
<thead>
<tr>
<th>Program Files</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Files</td>
<td>These files contain the source code for the program. They are '<em>.h', '</em>.cpp', '<em>.md', '</em>.rb', '<em>.xml', and '</em>.txt' files. Most of the source code is part of a Microsoft Visual Studio 2010 solution.</td>
</tr>
<tr>
<td>source_code.zip</td>
<td>This contains the source code of the program. It also contains a README.md file (can be opened by any text editor) that details how the program environment is to be set up along with a description of the files and how to run the code.</td>
</tr>
<tr>
<td>connected_graph.rb</td>
<td>This file contains the ruby code that finds the largest connected component in the graph generated by the source code and outputs it to a file.</td>
</tr>
</tbody>
</table>