Fast image matching using ORB and Multi-Index Hashing

by

Pat C. Goebel

Undergraduate Honours Thesis
Faculty of Science (Computing Science)
University of Ontario Institute of Technology

Supervisor(s): Dr F. Z. Qureshi

Abstract

Fast image matching using ORB and Multi-Index Hashing

Pat C. Goebel

Undergraduate Honours Thesis

Faculty of Science (Computing Science)

University of Ontario Institute of Technology

2014

We have combined the Multi-Index Hashing technique with large binary descriptors to provide fast, accurate image matching in large datasets. We have found that 256 bit rotated BRIEF descriptors generated with the oriented FAST feature detector can be matched in sub-linear time. We have shown this technique to be effective for databases of up to 10 million codes generated from 25,000 images. This research shows that accurate matching of images is possible for large image datasets while maintaining speed and space efficiency by coupling natively generated binary image descriptors with the Multi-Index Hashing technique.

Contents

1 Introduction **1**

1.1 Thesis Statement 1

1.2 Background: Problem and Related Work 1

1.2.1 Motivation 2

1.2.2 Multi-Index Hashing 2

2 Methods **4**

2.1 Definitions 4

2.2 General System Overview 4

2.3 ORB Module 5

2.4 MIH library 6

2.5 Datasets 6

2.6 Empirical Testing 7

2.6.1 Performance 7

2.6.2 Code Distribution 8

2.6.3 Hamming Radii needed for k-NN 8

3 Results **10**

3.1 Query Performance 10

3.2 Code Distribution 11

3.3 Hamming Radii needed for k-NN 13

4 Discussion **16**

4.1 Query Performance 16

4.2 Code Distribution 17

4.3 Hamming Radii needed for k-NN 18

5 Conclusion **19**

Chapter 1

Introduction

1.1 Thesis Statement

We aim to explore the effectiveness of the Multi-Index Hashing technique [**?**] for the fast matching of large, binary, image descriptors in large databases for the purpose of fast, accurate image matching.

1.2 Background: Problem and Related Work

Image matching is an active area of research with wide ranging applications, but achieving good performance with respect to speed, space and accuracy is still a challenge. There has been growing interest in mapping image data onto compact binary codes for fast nearest neighbours search. Binary codes are storage efficient and comparisons require just a small number of machine instructions, such that millions of binary codes can be compared to another in less than a second. Binary codes can also be used as direct indices into hash tables which can provide constant time lookups. In spite of the efficiency and speed of using binary codes to represent image data, most image feature description is expressed as vectors of floating point numbers which must be compared using a measure of Euclidean distance. While there has been work on mapping these real valued descriptors to binary codes [?, ?] the issue remains that these schemes provide an imperfect mapping from real valued vectors to binary codes, and that to achieve a strong correlation between the Euclidean distance for the real valued codes and the Hamming distance for the binary codes, large binary code lengths must be used. While matching long binary codes is still quite fast using linear scanning, they have posed a problem in that up until recently it was inefficient to quickly match binary codes longer than 30 bits [?] using the binary codes as indices into hash tables. With the introduction of the MIH technique it is possible to exactly match codes longer than ~30 bits faster than linear scanning.

1.2.1 Motivation

The motivation behind this work is the ability for the MIH technique to enable the use of high performance binary descriptors such as Rotated BRIEF. Binary descriptors do have the advantage of being easy to compare because Hamming distance is easily computed and is efficient on computer systems, but current limitations significantly decrease the efficiency of using longer binary codes which can have better matching properties than popular Euclidean distance based descriptors such as SIFT and SURF [?]. With the MIH method of calculating Hamming space nearest neighbours we gain performance in matching long binary codes. This performance gain is helpful for the hashing techniques on real valued descriptors, but also enables the use of natively generated binary descriptors which do not suffer from the degradation of matching rates (See Figure 3.5) due to hashing from Euclidean to binary descriptors, are faster to generate, and allow better matching properties.

1.2.2 Multi-Index Hashing

The Multi-Index Hashing technique introduces a fast method of searching for exact k-nearest neighbours in hamming space which is also space efficient [?] . Previous solutions were using binary codes as direct indices into hash tables but this approach was not used for codes longer than 32 bits because the memory requirements are prohibitive [?]. While the direct hashing is fast for shorter binary codes, it is desirable to use longer codes as short codes do not preserve similarity as well.

In the research introducing the MIH technique, an emphasis is placed on the mathematical properties of the proposed data structure, but empirical analysis of the structure was carried out using 64 and 128 bit codes, generated using Locality Sensitive Hashing and Minimal Loss Hashing from SIFT and GIST descriptor datasets. Analysis of the performance properties of 256 bit codes was not included in the research so we sought to analyze the performance of MIH using 256 bit codes.

Chapter 2

Methods

2.1 Definitions

ORB Oriented FAST and rotated BRIEF

LSH Locality-sensitive hashing

MLH Minimal Loss Hashing

MIH Multi-Index Hashing

2.2 General System Overview

This thesis integrates an Oriented FAST and Rotated BRIEF (ORB) feature detection and description module with a Multi-Index Hashing library. Our design utilized the ORB feature detector and descriptor extractor to generate 256 bit feature descriptor codes. These codes were then used to populate the multi-index hashing (MIH) data structure. The populated MIH structure was then queried using the provided interface.

To generate our list of binary codes we iterate through images in a directory and for each image we preprocess it by converting it to grayscale and applying a gaussian blur with radius 7 to eliminate noise. After the image has been preprocessed it is passed to ORB to generate features, then binary descriptors. For each image these descriptors are appended to a list of descriptors. When all the images have been iterated through we then can use the full list of descriptors to pass to an instance of the MIH datastructure to populate it.

A facility was created to serialize and deserialize the generated descriptors as to avoid generating the descriptors for every test.

2.3 ORB Module

The ORB module is part of the OpenCV library [?]. ORB takes as an input one image file and outputs a list of binary descriptors extracted from that image. In this paper, binary descriptors are defined as binary codes which represent the salient features of the image they are produced from. Any image file that OpenCV can read and process can be passed to ORB. The ORB processes these images into binary descriptors by using two sub-components: The Oriented Fast Feature Detector (FD) component, followed by the Rotated BRIEF Descriptor Extractor (DE).

The FD uses corner detection with an intensity centroid orientation measure as well as a scale pyramid which are passed through a Harris corner filter to eliminate edge matches, producing corner descriptors that have scale and orientation information [?]. These features are then passed onto the DE. The DE takes the image, a keypoint and the feature information and computes a binary code which represents that feature [?].

The authors of the ORB module in OpenCV chose a descriptor length of 256 bits for the output of the DE and this defined the size of binary codes used for our experiments.

2.4 MIH library

The MIH technique works by splitting every code used to populate the structure into m disjoint substrings and using these substrings as indices into m different hash tables. This allows us to search many fewer buckets in the hash tables as we know that, given a query radius of r, any code within this radius from the query will have at least one substring which differs by at most $\lfloor \frac{r}{m} \rfloor$ bits. This provides sub-linear query times for large databases of large codes.

The MIH code was modified to remove dependencies on MATLAB and was compiled as a library to be linked to by our code. 256 bit binary codes were used to populate the MIH structure and performance tests were carried out by querying the MIH structure, varying the number of neighbours searched for and by varying the substring length used by MIH. Also included in the MIH library is a function for performing a linear scan of the provided codes.

2.5 Datasets

Experiments are conducted using three vision corpora: MIRFLICKR-25000 collection [?] , Caltech-256 Object Category Dataset [?] and the 102 Category Flower Dataset [?]. These image sets were used to generate rotated BRIEF descriptors for our experiments. The number of images and the number of descriptors generated using ORB on these datasets are listed in the following table:

Dataset	**Images**	**Descriptors**
MIRFLICKR-25000	25,000	10,818,284
256 Object Category	30,607	9,996,971
102 Category Flower	8,189	3,615,406

2.6 Empirical Testing

Several tests were carried out using our developed application. Testing was performed on all three vision corpora for every test. We tested the performance of querying of the MIH structure under various conditions, and also performed some statistical tests on the codes we generated. All performance tests were executed on a machine with 256GiB of RAM and two Intel® Xeon® E5-2670 CPUs with 20MiB of cache and running at 2.6GHz. The process priority of the tests were set to -8 (high priority) to minimize noise from system load, as the machine was being shared by multiple users. The MIH library is single threaded [?] and therefore only utilized one thread of execution on one of the CPUs during tests.

2.6.1 Performance

Testing of the performance of the MIH library was performed by measuring how long the call to the query function provided by the MIH library blocked. The performance of the query call was measured for various values of k and m for each of the datasets. It is prudent to note that it is not possible to change the value of m after the MIH structure is created, thus, for each value of m the MIH structure must be recreated and repopulated. It is possible to change the value of k on the fly so this lead us to create and populate an MIH structure with a specific m, query it for various k nearest neighbours, then move on to the next value of m.

For each dataset we first loaded the descriptors into memory by deserializing them from a file stored on the hard disk into an OpenCV Mat structure. The Mat structure contains information about the codes such as the number of codes and their length, as well as a pointer to the codes in contiguous memory. For each dataset we selected a random image from that dataset and computed binary descriptors for it in the same manner as for the creation of the binary codes for that dataset. We stored these descriptors in a separate Mat structure. Each image yielded ~400 binary descriptors.

We populate MIH by calling the `populate` function with a pointer to the codes, the number of codes in the list and the length of the codes in bits.

For the query we record the processes current cpu time in nanoseconds using the systems realtime clock and then we call the `batchquery` function, passing in a pointer to the query codes, the number of query codes and the length of the codes in bits, as well as various arrays used to store the results of the query. After the function has returned we record the process cpu time again and print out the difference to the console.

When plotting this data we divide the time it took to run `batchquery` by the number of descriptors supplied to it in the query. This gives us an average measure of the time required to query each descriptor.

2.6.2 Code Distribution

Statistics on the distribution of binary codes in each of the datasets were calculated by storing a count for each of 256 bits and going through all the codes, incrementing the count for each bit in each code. After going through all the codes the count for each bit was divided by the total number of codes processed to give us a mean value for each of the 256 bits of the codes. A χ^2 test was set up for each dataset to test if each bit in all the codes were uniformly distributed. Each bit was modeled as a sample that was put into two bins, 0 and 1. Thresholds for statistical significant deviation from the uniform distribution were calculated based on this χ^2 test.

2.6.3 Hamming Radii needed for k-NN

Statistics on the Hammming radius needed for each code to find it's k nearest neighbours was calculated for the first 10000 codes in each dataset by taking each code and querying it's k nearest neighbours and finding the neighbour with the largest Hamming distance. A count was kept for each possible distance (0 to 256) and after all the codes had been queried, every count was divided by the number of codes queried to give the proportion of codes that required that radius.

Chapter 3

Results

3.1 Query Performance

From the data gathered for the performance of the queries plotted in Figure 3.1 it can be shown that even with 256 bit codes we see sub-linear query times for the larger datasets. It is also evident that these speedups are more pronounced with larger database sizes. The observed empirical optimal values of m are close to but do not completely agree with the equation $b/\log_2 n$ proposed in [?].

Dataset	**Descriptors**	**Calculated Optimal** m	**Empirical Optimal** m
MIRFLICKR-25000	10818284	10.96	10
256 Object Category	9996971	11.01	11
102 Category Flower	3615406	11.75	14

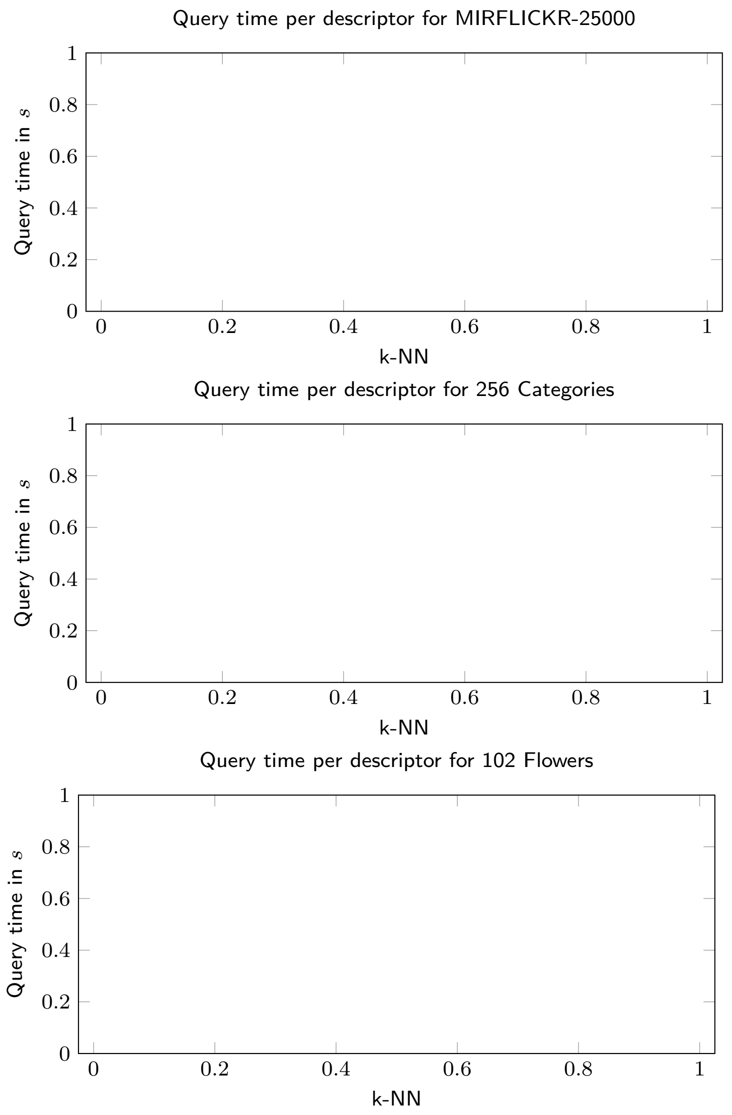


Figure 3.1: Query time per descriptor *vs* k for various m, for each of the three vision corpora used.

3.2 Code Distribution

From the data gathered for the distribution of binary codes plotted in Figure 3.2 we can see that the bits are not uniformly distributed as there are almost no bits which passed a χ^2 test against a uniform distribution for $p = 0.05$ (bits that passed are marked in green). The assumption of uniformly distributed codes is made when calculating the theoretical performance of the MIH structure. We can also see that there is a pattern in the expected values of the bits between the 3 different datasets.

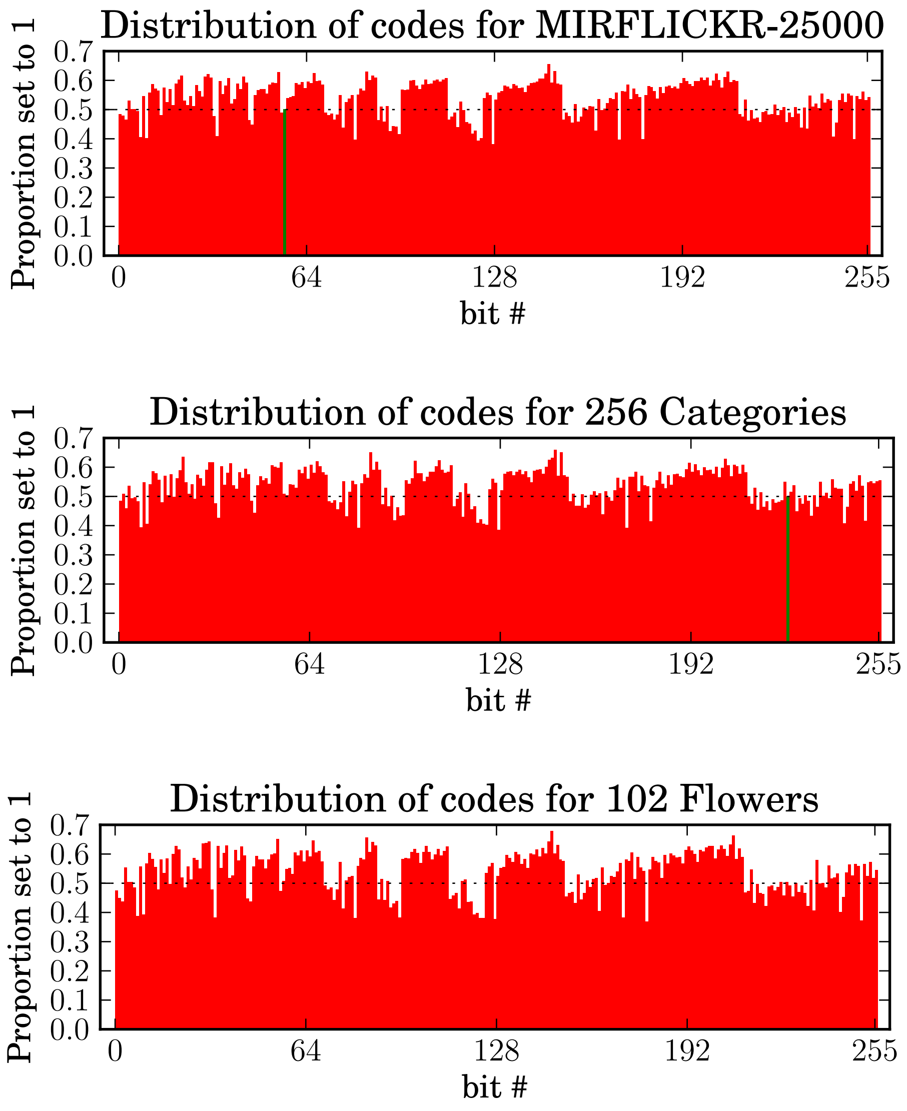


Figure 3.2: Empirical expected value for each bit for all codes in each of the three vision corpora used.

Text below distribution

3.3 Hamming Radii needed for k-NN

We can see from the Hamming radii required for $k = 10$ and $k = 1000$ nearest neighbours for our three datasets that the requred radii are much larger in our 256 bit codes compared to the 64 and 128 bit codes in Figure 3.4. The deviation of the required radii are also much higher for our natively generated 256 bit codes.

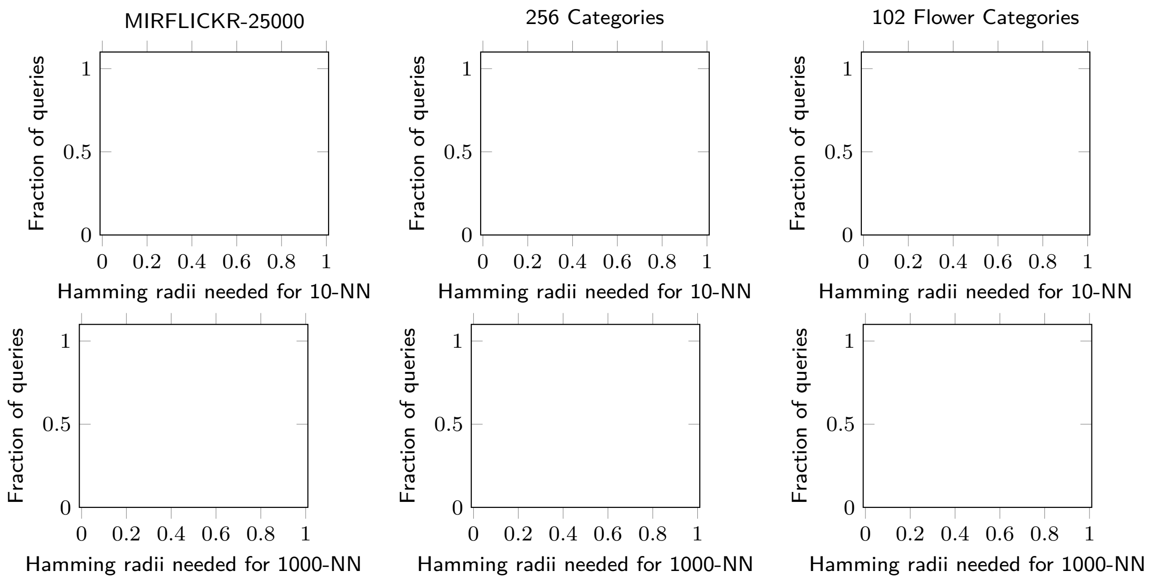


Figure 3.3: Shown are the histograms of the search radii that are required to find 10-NN and 1000-NN for 256 bit codes generated using ORB from the three datasets used.

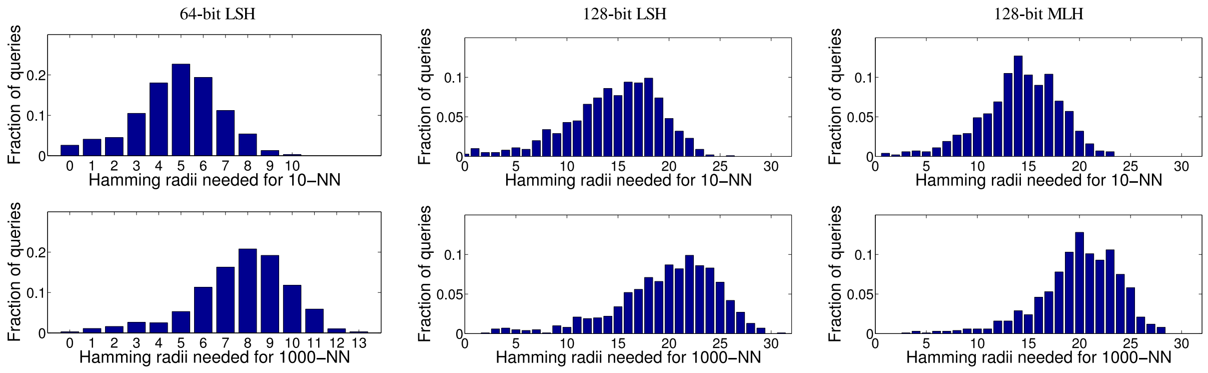
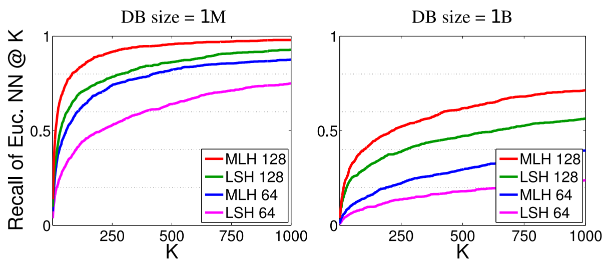


Figure 3.4: From Norouzi *et al.* [?]: Shown are histograms of the search radii that are required to find 10-NN and 1000-NN, for 64 and 128-bit code from LSH [?], and 128-bit codes from MLH [?], based on 1B SIFT descriptors [?]. Clearly shown are the relatively large search radii required for both the 10-NN and the 1000-NN tasks, as well as the increase in the radii required when using 128 bits versus 64 bits.

Figure 3.5: From Norouzi *et al.* [**?**]: Recall rates for BIGANN dataset [**?**] (1M and 1B sub-sets) obtained by K-NN on 64- and 128-bit MLH and LSH codes.



Chapter 4

Discussion

While the results from our testing were encouraging as to the viability of using natively generated 256 bit binary codes in a MIH structure, some discussion is warranted to explore the patterns and anomalies in the results that may elucidate further research.

4.1 Query Performance

The performance tests show that our queries were running in sub-linear time up to 1000 nearest neighbours for the larger databases. It is interesting to note that for the 102 Flowers database the queries were only faster than linear scanning before k reached around 250. This seems to indicate that the margin between MIH and linear scanning in terms of the query speed will increase with the database size, which is a nice result.

Also interesting is how the optimal m values were not quite as predicted. For the MIRFLICKR-25000 dataset, $m = 10$ and $m = 11$ were very close in terms of performance but $m = 10$ was faster and the equation given by [?] predicted an m value closer to 11. The 256 Object Category dataset fit nicely with the predicted m value. The 102 Flowers dataset was very far from the predicted optimal m, but it seems so far that the smaller dataset did not perform well with MIH.

It is important to note that query times were many times lower than linear scanning for small values of k, which means that for applications only requiring a small number of nearest neighbours MIH could be an effective technique even for small datasets. With each image having hundreds of descriptors it may not be necessary to get a large number of nearest neighbours for each descriptor to get a statistically significant match, especially since the codes have not been through a hashing function and are true nearest neighbours.

Overall, the performance of the system was good when not querying for large numbers of nearest neighbours. This could be an effect of having small datasets as with larger datasets of shorter codes, the Hamming radii required to get k nearest neighbours is lower, resulting in less lookups.

4.2 Code Distribution

The calculation of the code distribution turned up the most unexpected result in all our tests. It was hypothesized that the descriptors would have a relatively uniform distribution but this turned out to be untrue. After setting up χ^2 statistical tests for the bits in each dataset we found that very few of the bits could be considered uniformly distributed. What was even more interesting than that was that the codes in each dataset appeared to follow a pattern as to the mean value of each bit. This can be seen visually in our histogram of the bits in the codes. Since our datasets are fairly different in content, this pattern suggests that it is an artifact of the ORB module. More research into the methods of ORB and more statistical analyses of the codes generated may lead to smaller and better performing binary descriptors.

As well, the calculations performed in [**?**] relied upon the assumption that the codes that were in the database were uniformly distributed. This suggests that further analysis is required to determine the effects that different distributions of codes have on the MIH query performance as descriptors from working datasets may evidently vary significantly from uniformly distributed.

4.3 Hamming Radii needed for k-NN

The Hamming radii needed for k-NN for our three datasets have relatively large means and deviations compared to those of the 64 and 128 bit datasets used in [**?**]. I believe that this manifests itself as lowered performance for larger values of k, as we must search more buckets for codes that fall within k-NN. While with uniformly distributed codes the mean radii we have to search will go up with the code length it would interesting to see how the theoretical average radii compare to the empirical radii for the different dataset sizes and descriptor extraction techniques as these could indicate why the empirical optimal number of substrings m did not always match the predicted optimal m.

Chapter 5

Conclusion

We have found that it is possible to quickly, efficiently and accurately match large binary codes in large databases of codes generated through ORB to perform fast image matching. This technique improves upon using hashing techniques on real valued descriptors to generate binary codes as with larger datasets these hashing techniques lose effectiveness in mapping real codes to binary codes, while binary codes generated by ORB do not suffer this degradation. As well this technique allows matching of these long binary descriptors in sub-linear times in respect to the database size. The Multi-Index hashing technique ensures that as database sizes grow the query time for these databases will increase with the square root of the size of the databases while using ORB ensures that as the database size grows the matching effectiveness of the codes will not drastically decrease. This research shows that there is promise in using natively generated binary descriptors for matching images quickly and that with current technology this technique can compete with current effective techniques.