University of Alberta

APPLICATION OF LOCALITY SENSITIVE HASHING TO FEATURE MATCHING AND LOOP CLOSURE DETECTION

by

Hossein Shahbazi

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Abstract

My thesis focuses on automatic parameter selection for euclidean distance version of Locality Sensitive Hashing (LSH) and solving visual loop closure detection by using LSH. LSH is a class of functions for probabilistic nearest neighbor search. Although some work has been done for parameter selection of LSH, having three parameters and lack of guarantees on the running time, restricts the usage of LSH. We propose a method for finding optimal LSH parameters when data distribution meets certain properties.

Loop closure detection is the problem of deciding whether a robot has visited its current location before. This problem arises in both metric and visual SLAM (Simultaneous Localization and Mapping) applications and it is crucial for creating consistent maps. In our approach, we use hashing to efficiently find similar visual features. This enables us to detect loop closures in real-time without the need to pre-process the data as is the case with the Bag-of-Words (BOW) approach.

We evaluate our parameter selection and loop closure detection methods by running experiments on real world and synthetic data. To show the effectiveness of our loop closure detection approach, we compare the running time and precision-recalls for our method and the BOW approach coupled with direct feature matching. Our approach has higher recall for the same precision in both sets of our experiments. The running time of our LSH system is comparable to the time that is required for extracting SIFT (Scale Invariant Feature Transform) features and is suitable for real-time applications.

Acknowledgements

I would like to thank my supervisor, Hong Zhang for his support and level of involvement. I never felt left alone nor too much pressured with my research.

I would also like to thank Kiana Hajebi, who helped me with capturing datasets and provided invaluable feedback on my work.

I thank all my colleagues and friends at University of Alberta who helped me during these two years and made my time fun and worthwhile.

Last but not least, I thank my family who have always been there for me.

Table of Contents

1 Introduction **1**
- 1.1 Outline of the Problem . . . 1
- 1.2 Thesis Overview . . . 2

2 Background **3**
- 2.1 SLAM: Simultaneous Localization and Mapping . . . 3
  - 2.1.1 Metric SLAM: EKF SLAM and FastSLAM . . . 4
  - 2.1.2 Visual SLAM . . . 4
  - 2.1.3 Loop Closure Detection . . . 5
- 2.2 CBIR: Content-Based Image Retrieval . . . 5
  - 2.2.1 Local Visual Features . . . 6
  - 2.2.2 The Distance Ratio Technique . . . 6
  - 2.2.3 Multi-View Geometric Verification . . . 7
- 2.3 LSH: Locality Sensitive Hashing . . . 8
  - 2.3.1 Motivation . . . 8
  - 2.3.2 Introduction . . . 9
  - 2.3.3 Improvements on LSH . . . 11
  - 2.3.4 Discussion . . . 13
  - 2.3.5 E2LSH Parameter Setting . . . 14

3 Optimal Selection of LSH Parameters **15**
- 3.1 Calculating the Running Time . . . 15
- 3.2 Computation of Collision Probability for the Weak Hash Functions . . . 16
- 3.3 Distance Distributions . . . 20
- 3.4 Expected Selectivity of Hash Functions . . . 25
- 3.5 Distance Threshold for SIFT Features . . . 27
- 3.6 Solving for Optimal Parameters . . . 29
- 3.7 Conclusions . . . 30

4 Experiments **33**
- 4.1 Datasets . . . 33
- 4.2 Evaluation of Selectivity Prediction Method . . . 34
  - 4.2.1 Experiment Setup . . . 35
  - 4.2.2 Results and Conclusion . . . 35
- 4.3 E2LSH Parameter Settings . . . 36
  - 4.3.1 Experiment Setup . . . 37
  - 4.3.2 Experiment Results and Conclusion . . . 37
- 4.4 Loop Closure Detection Experiments . . . 38
  - 4.4.1 The Bag-of-Words Approach . . . 39
  - 4.4.2 E2LSH . . . 40
  - 4.4.3 Results and Conclusion . . . 41

5 Conclusions and Future Work **46**

Bibliography **47**

A Appendix 1: Plots of E2LSH using Bounds on the Sphere Caps **49**

List of Tables

3.1 Discrepancy of overall distance distributions in different SIFT datasets 25
3.2 Homogeneity Index of Random and sample SIFT datasets 25

4.1 Empirical and Estimated selectivity of E2LSH structures 36
4.2 Empirical and Estimated selectivity of IE2LSH structures 36
4.3 Results of NNS on SIFT features 38
4.4 Demonstration of distance ratio for matching multiple features. Distances and distance ratios for the points shown in Figure 4.4. 41
4.5 Running Times for the City Center dataset 41
4.6 Running Times for the Google Pits side view dataset 42

List of Figures

2.1 Probability of correct / incorrect match based on distance ratio. 7

3.1 Demonstration of computation of probability of collision for a single hash function. The vector represents the function and the blue lines mark the margins of the bins. The ratio of the length of the red parts to the circumference of the circle is the probability of collision. 17
3.2 Sphere cap at distance u from sphere center. For demonstration of Equation 3.8. . 18
3.3 Ratio of cap surface area to sphere surface as a function of u in different dimensions. As the dimensionality increases, the surface area of caps decreases more sharply as the base of the cap moves away from the center of the hyper-sphere 18
3.4 Actual ratio of cap surface vs. ratio computed from bounds. Both ratios get very close to 0 quickly. 19
3.5 Probability of collision of two points in a randomly selected hash function h as a function of $\frac{W}{r}$. Only small values of W are of interest to us. Values larger than 1 definitely lead to infeasible running times. 20
3.6 Example demonstrating relative distance distributions. The gray disk in (a) shows the distribution of points in space. (b) and (c) show DD_{P_1} and CDD_{P_1} respectively. The RDDs for all the points on the red circle are the same as DD_{P_1}. 21
3.7 Distance distributions of SIFT features. Figure (a) shows 10 sample SIFT RDDs to show the amount of variations in them. Figures (b) and (c) show the DD and CDD of SIFT features with their standard deviations. 23
3.8 The overall distance distribution of different SIFT datasets. The distance distribution of SIFT features is fairly stable in different datasets (see Table 3.1). 24
3.9 Distance distributions of random vectors. Figures (a) and (b) show the DD and CDD of random generated vectors with their standard deviations. 25
3.10 Expected selectivity of weak hash functions h on uniformly distributed points and SIFT features as a function of W. As W increases, $E[S_h]$ approaches 1. We are interested in small W values below 0.3 which is the distance at which the SIFT features match. The selectivity for SIFT features is very close to that of the randomly generated points. 26
3.11 Overall distance distribution of matched and unmatched SIFT features. The little overlap between the plots shows the euclidean distance is a reasonable choice for matching SIFT features. 28
3.12 The (a) success probability, (b) running time and (c) L optimal value as functions of W and K for E2LSH. The time limit was set to 400K operations (3K vector dot products). The maximum number of tables was set to 200 (roughly 3GB). 31
3.13 The (a) success probability, (b) running time and (c) L optimal value as functions of W and K for IE2LSH. The time limit was set to 400K operations (3K vector dot products). The maximum number of tables was set to 200 (roughly 3GB) which implies that $m \leq 21$. 32

4.1 Sample images from the City Center dataset. The images are not calibrated. 34
4.2 Sample images from the GPS dataset. The images are not calibrated and for each location, there are two images looking sideways from the vehicle trajectory. 34
4.3 Overview of our loop closure detection system. 39
4.4 Demonstration of distance ratio for matching multiple features. Blue points show the points in dataset and the red point is the query point. 41
4.5 Precision-recall curves for (a) the City Center dataset and (b) the Google Pits side view dataset. The points in each curve correspond to different thresholds for accepting loop closing images. 43

4.6 Map of the City Center dataset with the loop closures drawn. The red lines are the actual loop closures and the green lines are the loop closures found by the LSH DR 2 configuration of our algorithm. 198 loop closures are detected out of around 1K loop closures. 44
4.7 Map of the Google Pits dataset with the loop closures drawn. The red lines are the actual loop closures and the green lines are the loop closures found by the LSH DR 2 configuration of our algorithm. 45

A.1 The (a) success probability, (b) running time and (c) L optimal value as functions of W and K for E2LSH. The time limit was set to 400K operations (3K vector dot products). The maximum number of tables was set to 200 (roughly 3GB). 50
A.2 The (a) success probability, (b) running time and (c) L optimal value as functions of W and K for IE2LSH. The time limit was set to 400K operations (3K vector dot products). The maximum number of tables was set to 200 (roughly 3GB) which implies that $m \leq 21$. 51
...

List of Abbreviations

SLAM:	Simultaneous Localization and Mapping
SIFT:	Scale Invariant Feature Transform
SURF:	Speeded Up Robust Features
BRIEF:	Binary Robust Independent Elementary Features
LSH:	Locality Sensitive Hashing
E2LSH:	Euclidean Distance Locality Sensitive Hashing
IE2LSH:	Improved E2LSH
DD:	Distance Distribution
CDD:	Cumulative Distance Distribution
RDD:	Relative (local) Distance Distribution
CRDD:	Relative Cumulative Distance Distribution
GPP dataset:	Google Pittsburgh Panorama Dataset
GPS dataset:	Google Pittsburgh Side Dataset
CC dataset:	City Center Dataset
BOW:	Bag of Words

List of Symbols

S^d: surface area of a hyper-sphere in d dimensions.
S_u^d: cap surface area of a hyper-sphere in d dimensions at distance u from the center.
$\mathcal{H}$: a family of weak hash functions.
$\mathcal{G}$: a family of generalized hash functions.
$sel(h)$ or S_h: selectivity of hash function h
$S_{L,K}$: selectivity of an E2LSH hash structure with parameters L and K
$S_{m,K}$: selectivity of an IE2LSH hash structure with parameters m and K
P_{col}^h: probability of collision of two random points from data distribution in hash function h.

Chapter 1

Introduction

Autonomous robot navigation has been one of the main focuses of the robotics community. The ultimate goal of this field is to develop robots that are capable of detecting their location in their environment and reaching a goal location without human interference. Despite three decades of research in this area, the applications of navigational robots remain limited to very small and simple environments.

The specific problem of creating a map of an environment while keeping track of the position in the map is called Simultaneous Localization and Mapping (SLAM). It has been the main focus of research in mobile robotics. The SLAM problem can be dealt with in different frameworks with different techniques.

In this thesis, we try to address one of the fundamental problems of SLAM, namely the *Loop Closure Detection* (LCD) problem. It is the problem of deciding whether a given location has been visited by the robot in the past. Reliable LCD is essential for creating consistent maps and solving the SLAM problem. More specifically, this thesis is centered around detecting loop closures in visual SLAM, where the locations in the map are represented by images and no metric information is available. In Section 1.1 we will describe the problem in a concrete way and then in Section 1.2 we overview the thesis.

1.1 Outline of the Problem

The problem that we tackle in this thesis is as follows: given an image representing a location, is that location present in the map and if so, which location does it correspond to?

This problem, known as visual LCD, is not trivial. Firstly the changes in the environment and moving objects cause the images of the same location taken at different times to look different. The lighting and weather conditions also change the appearance of environment over time. Secondly, the images are not taken exactly from the same viewpoint every time. The robot usually has a small displacement when coming back to a location and we should be able to cope with small viewpoint changes. Finally, there are similarly looking locations in the environment. For example, some patterns like brick walls, roads and cars, trees, *etc.* may appear in multiple locations. This problem is known as *Perceptual Aliasing*.

Apart from the correctness perspective, the efficiency of the algorithms is also very important. For each location, the time spent on detecting loop closures should be in order of seconds; with very large maps that contain many thousands of images, satisfying the time constraint is really hard.

We address the visual LCD problem by reducing it to image comparison. By computing the similarity of the query image with every image in the map and thresholding that similarity, one can find loop closing images. We perform this computation efficiently by combining fast nearest neighbor search algorithms and state-of-the-art techniques from the vision community.

We have not dealt with the uncertainty aspect present in SLAM. We focus on computing the image similarities in a fast and accurate way. Probabilistic frameworks like Fabmap [12] can be wrapped around any method that can generate a sorted small candidate list of locations for LCD. One can use such frameworks for dealing with the uncertainty and perceptual aliasing and it is clear that with a more accurate and efficient method of computing similarities, the overall performance of the system will improve.

In this thesis, we make two contributions:

- Our approach performs better than current Bag of Words approach coupled with direct feature matching in terms of loop closure detection recall in the datasets we have used in our experiments.

- We give detailed instructions on how to set the parameters of E2LSH algorithm given the constraints on the running time, space requirements or performance of the algorithm. Our work goes beyond what is present in [14] which guarantees a lower bound on the probability of success of the algorithm.

1.2 Thesis Overview

The thesis is structured as follows. In Chapter 2 we present some topics that are related to our work from mobile robotics, computer vision and locality sensitive hashing. In Chapter 3 we discuss in detail our approach and the contributions we have made. In Chapter 4 we explain the experiments for comparing our method with previously developed methods and their results. Finally, the conclusions and some potential directions for further investigation are presented in Chapter 5.

Chapter 2

Background

In this chapter we present background topics related to the thesis. The topics reside in three fields: Simultaneous Localization and Mapping (SLAM), Image Retrieval and Locality Sensitive Hashing. Sections 2.1, 2.2 and 2.3 explain these topics respectively.

2.1 SLAM: Simultaneous Localization and Mapping

In mobile robotics, it is critical for a robot to recognize its pose within its environment. SLAM is the process by which a robot creates a map of its environment and deduces its pose in that map at the same time. Both localization and mapping are challenging problems because of the errors that occur in sensor readings and robot's movement. The sensor readings are subject to noise and bias and the motion model is just an estimate of the robot's actual movement. As a result of these errors, SLAM frameworks have to tackle the problem in a probabilistic way.

The common method of handling uncertainty in the context of SLAM is by using bayesian inference [29] [12]. If robot's observation at time step k is represented by Z_k , $Z^k = \{Z_1, Z_2, ..., Z_k\}$ and the robot pose (position and orientation) is denoted by x, x is estimated by:

$$P(x|Z^k) = \frac{P(Z_k|x)P(x|Z^{k-1})}{P(Z_k|Z^{k-1})} \tag{2.1}$$

The robot observation depends on the types of sensors the robot uses. It can be odometery information, sonar or laser range data, images from camera, *etc.* . The robot pose (x) depends on the internal representation of the map. It can be x, y, z Cartesian coordinates, the index to a discrete set of locations, *etc.* .

There are two general strands of SLAM algorithms: metric SLAM and Visual SLAM. In metric SLAM, the landmarks and the robot's pose are estimated in a single global coordinate system. In visual SLAM, the map is topological and contains no metric information. Depending on the task at hand, one type of SLAM might be more suitable. We will describe the major works in both metric and visual SLAM in the subsequent subsections.

2.1.1 Metric SLAM: EKF SLAM and FastSLAM

EKF SLAM was introduced in [29] and used extensively afterwards. In EKF SLAM, the world is represented by a set of landmarks and the pose of the robot and the landmarks are estimated by Gaussian distributions. While the EKF formulation was successful, it had some problems. First, the assumption that the poses can be modeled by Gaussian distributions does not hold in all cases. There are cases where motion model is bimodal or the error in angular velocity is high. In such cases, EKF SLAM is insufficient and may generate an inconsistent map. Secondly, the size of the covariance matrix of robot and landmark poses grows quadratically with the size of the map. Therefore, the algorithm becomes intractable as the number of landmarks passes a few hundreds. By using submaps as in [7], this problem can be alleviated. However, as a general observation, the convergence rate decreases as the number of submaps increases. Also, one needs to estimate the relative transformations between submaps in order to perform navigation. Finally, EKF SLAM introduces linearization errors. The Kalman Filter itself is only applicable to linear processes. In EKF SLAM, the functions are estimated by their values around the current estimates and this is a source of error.

An alternative to EKF SLAM is known as FastSLAM based on particle filter implementation of the Bayes filter (Equation 2.1). FastSLAM relies on the fact that landmark positions are independent given the robot path. In FastSLAM, the robot pose is represented by a set of particles and each particle keeps track of landmarks independently. Because there are no assumptions about the distribution of poses, FastSLAM can work with nonlinear processes or non-guassian distributions. However, in FastSLAM it is hard to maintain a diverse set of particles and the particles tend to converge to the most likely positions of the robot. Therefore, the systems that are based on FastSLAM have difficulty when the robot trajectory is long.

2.1.2 Visual SLAM

While metric SLAM has been the dominant approach to SLAM, visual SLAM has been gaining more popularity in the robotics community during the past decade. In visual SLAM, the robot locations are characterized by images and the robot map is topological, showing the connections between locations. While a topological map does not contain any metric information, in some cases it is adequate for successful navigation [5]. If an algorithm needs metric information, it is possible to embed some metric information into the topological map for easier navigation [28]. Such systems are sometimes called hybrid SLAM because they benefit from both types of information.

FABMap [12] is an example of a visual SLAM system that can perform well in real-world environments. It is able to detect loop closures and build a consistent map in large scale maps (10000+ locations). FABMap uses the bayesian framework to assess the probability of loop closures. Using the probabilistic framework helps avoid false positive detection, *i.e.,* mistakenly detecting loop closures when there are similarly looking locations in the environment.

2.1.3 Loop Closure Detection

There are various problems that are subject of current research in SLAM. Loop Closure Detection (LCD) is one of the most important problems. It is the problem of deciding whether the current view of the robot is from an existing location in the robot map or it is from a new location. In case the current view belongs to a location in the map, we are interested in knowing which location. If a robot fails at detecting true loop closures, there will be duplicate locations in the map for the same external location. On the other hand, if the robot generates false positives, the map will contain one node for multiple distinct locations in the environment. Both cases will cause the robot map to become inconsistent. However, the problems cause by false positives are more drastic. False positives are handled vigorously in current visual SLAM algorithms by applying a strict and computationally costly verification step that is based on mullti-view geometry. As a result, for practical purposes, false positives can be effectively eliminated and the 100% precision can be achieved. On the other hand, false negatives are less tricky to handle. In the case of false negatives, the missed loop closing locations could be aligned to their correct positions by the nearby correctly detected loop closures.

2.2 CBIR: Content-Based Image Retrieval

CBIR deals with the problem of finding similar images in a large database. Only pixel information of the images (textures, shapes, colors, etc.) is used and annotations, tags, keywords and other extra information are ignored. CBIR has applications in many fields; medical imaging, web search, visual SLAM to name a few.

In LCD, the aim is to determine whether a scene has been visited before. In such cases, we expect the view of the robot to be fairly the same. Therefore, a reliable measure of similarity between images can be used in solving the visual LCD problem. The first methods for image similarity represented the whole image as a single feature and compared the entire images. Pixelwise image difference and global histograms are examples of such techniques. The main problem with these techniques is that they are sensitive to local changes in images like moving objects, changes in lighting, etc. A better approach for computing similarities is the use of local features. By representing an image as a set of local features, image matching can be carried out reliably because the local features can be detected and used as long as the objects they correspond to are visible in the image. There are local features that are invariant to affine transformations, changes in lighting and projection. Such features are shown to allow reliable image matching [33]. In this section, we will overview some types of local visual features and also the techniques that can be used to find similar images in image databases.

2.2.1 Local Visual Features

To use visual features for describing an image, a feature detector and a feature descriptor are needed. Feature detector selects the points in the image that are more useful for feature matching. Harris Corner detector, SIFT [21] and SURF (Speeded Up Robust Features) [3] are some of the most common feature detectors. After an interest point has been found, a feature descriptor is created to capture the information on the region of the image. We are interested in visual features that are invariant to rotation, scaling and 3D projection.

SIFT features have been used in many works [26][25] and proved to be accurate for feature matching. SIFT feature detector uses difference of Gaussians function in scale and space to find points in the image that are maximum or minimum in their neighborhood. The descriptor computes image gradients in a small area around the interest point and stores the number of gradient vectors in each direction in the feature descriptor after weighting and normalization. Each feature is consisted of 16 bins of 8 numbers (128 dimensions). The direction with most local gradient vectors is selected as the feature direction. This direction makes SIFT features prune to image-plane rotations.

Among the visual features, SIFT features can be matched more reliably and therefore are more suitable for our application [30][23]. The drawback of using SIFT features is the extraction procedure. For an image that has around 300 SIFT features, the time spent on feature extraction can be up to half a second on an ordinary machine. Researchers have tried to overcome the slowness of the SIFT features in many works. In [30], the authors compress the SIFT features trying to preserve their distance and use the reduced features for matching. Their results show that the repeatability of the reduced features is comparable to that of the original SIFT features. However we cannot use their method because their approach needs the SIFT features to run an optimization on them and find an optimal projection matrix. Other approaches that try to create SIFT-like features with smaller sizes or more efficient implementations such as BRIEF (Binary Robust Independent Elementary Features) [6] and SURF [3] are less accurate for feature matching[30]. We use SIFT features in our experiments; However, we mainly focus on nearest neighbor search aspect of our approach. There might be more suitable visual features for our task.

2.2.2 The Distance Ratio Technique

Once we have described images in terms of their visual features, we need a method to compare two images based on these. Since each image is represented by a set of visual features, we can use Jaccard Coefficient as a measure of the similarity of two images. The similarity of image I_A and I_B is computed by:

$$S = \frac{|A \cap B|}{|A \cup B|} \tag{2.2}$$

A and B are the sets of features of images I_A and I_B respectively. To use the above equation, a method for detecting matching features is needed. The simplest method would be to threshold

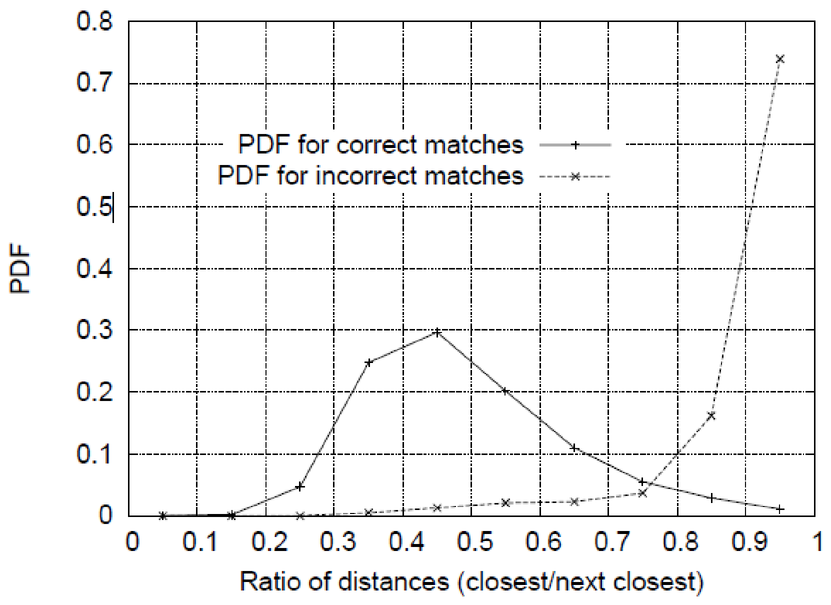


Figure 2.1: Probability of correct / incorrect match based on distance ratio.

the euclidean or manhattan distance between the SIFT features. Lowe *et al.* [20] use a different technique to match the SIFT features. The distances between features depends on the lighting conditions, image noise and 3D transformations and by using their ratios, the matching method will be less vulnerable to these variations[23]. The distance ratio is defined for features of one image when we are matching the features of a pair of images. For a single feature $a_i \in A$, the distance ratio is given by the ratio of its distance to closest feature from I_B, $b_c \in B$, to the distance to the second closest feature from I_B:

$$\delta(a_i) = \frac{d(a_i, b_c)}{\min_{b_j \in B - \{b_c\}}(a_i, b_j)} \tag{2.3}$$

Figure 2.1[1] shows the probability of correct and incorrect matches based on the distance ratio[20]. By picking a distance ratio of 0.8 for example, it is possible to prune 90 percent of incorrect matches while missing less than 5 percent of the correct matches.

The distance ratio technique relies on the fact that each feature from I_A can match at most one of the features of I_B. It cannot be used to match a feature with multiple features in different images or when the objects in the scene are self-similar.

2.2.3 Multi-View Geometric Verification

Although feature matching is very accurate, it is possible to obtain even more measures of similarity by taking into account the relative position of features with respect to each other. By considering the underlying 3D location of visual features, one can relate the image location of the features using

[1]Figure taken from [20]

the Fundamental matrix:

$$x_1 F x_2 = 0 \tag{2.4}$$

x_1 and x_2 are local position of two features and F is the fundamental matrix between the poses of the camera. This additional step, known as Multi View Geometry (MVG) verification has been used in many works [13] [9]. After computing the fundamental matrix, we can take the percentage of matched features that satisfy the MVG constraints as the similarity of the two images. Computing the best fundamental matrix is however computationally expensive and it is possible to only check a few (under 100) pairs of images in each second using MVG verification.

2.3 LSH: Locality Sensitive Hashing

LSH is a method for approximate nearest neighbor search (NNS). The main idea of LSH is to hash data points in a way that close points are more likely to collide (being hashed to the same value). By computing the hash value for a query point and looking up its hash bucket, we can get a set of candidate points that are likely to be close to the query point. LSH is known to perform better than other NNS methods in high dimensional spaces.In the remainder of this chapter, we present detailed information about LSH and various function families that have been developed over the past decade. We then tailor LSH for our application in matching SIFT features and detecting loop closures.

2.3.1 Motivation

Similarity search has become important in data mining applications such as content-based image, video and sound retrieval. These objects are either represented or characterized by vectors in high dimensions and hence, similarity search is usually carried out by *nearest neighbor search* (NNS) in high dimensional spaces.

One related application of nearest neighbor search is in feature matching. For example in the distance ratio technique, one needs to find the first two nearest neighbors of a feature. In visual word quantization of visual features, one needs to find the visual word that is closest to the query feature.

There are different approaches for nearest neighbor search. Tree based methods such as KD-Trees [4] and R-Trees [17] are known to perform worse than exhaustive search when dimensionality of the data exceeds a few dozens. The work of Weber *et al.* [32] states that all space-partitioning NNS methods, including the tree-based partitioning algorithms, will eventually degrade to exhaustive search as the dimensionality of the space increases.

Locality sensitive hashing finds near neighbors by hashing high dimensional vectors so that closer vectors are more likely to collide. It has been shown to be successful for high dimensional nearest neighbor search. In [8] the min-hash variant of LSH algorithm has been used in conjunction with the BOW quantization of the visual features for detecting near duplicate images.

2.3.2 Introduction

Let $\mathbf{S}$ be a set and $\mathbf{D}(.,.)$ a distance metric on the elements of $\mathbf{S}$. As originally introduced in [18], a family of functions $\mathcal{H}$ is called (r_1, r_2, p_1, p_2)-sensitive if for any $p, q \in S$:

$$\text{if } D(p,q) < r_1 \text{ then } Pr_{\mathcal{H}}\left[h(p) = h(q)\right] \geq p_1$$
$$\text{if } D(p,q) > r_2 \text{ then } Pr_{\mathcal{H}}\left[h(p) = h(q)\right] \leq p_2$$

A family is of interest only when $p_1 > p_2$ and $r_1 < r_2$. It is in this case that near points are more likely to collide than the far points. The discriminative power of a hash function $h_i \in \mathcal{H}$ is measured by its $selectivity$. Selectivity of a hash function h is the average ratio of the number of points it prunes. Multiplying selectivity by the number of points in the dataset gives the expected number of points returned by that function.

$$sel(h) = \frac{E[n]}{N}$$

n is the number of returned candidates by the hash function and N is the size of the data set. Note that selectivity is not an intrinsic characteristic of the hash functions and depends on the dataset that is being used and the distribution of the query points.

Assuming that we have a weak family $\mathcal{H}$ of locality sensitive functions, it is possible to create a second family $\mathcal{G}$ with higher discriminative power at the cost of a higher hashing time. The method is generic and applicable to any function family $\mathcal{H}$.

The process of creating $g_j \in \mathcal{G}$ is as follows. We select K functions $h_{j,1}$, ..., $h_{j,K}$ randomly from $\mathcal{H}$ and let $g_j(v)$ be the concatenation of the outputs of these functions:

$$g_j(v) = (h_{j,1}(v), ..., h_{j,K}(v))$$

To create g functions we need to set two parameters: the number of base functions (h_i) to concatenate (k) and the number of functions to use (L). Considering that h_is are selected at random in each g, the probability of collision of near points (points at distance r_1 or closer) on $g \in \mathcal{G}$ is more than p_1^K. The probability of collision of far points (points at distance r_2 or farther) is smaller than p_2^K. Therefore, as we increase K, both probabilities decrease, but p_2^K decreases with a faster rate as $p_1 > p_2$.

To process a query point q, the point is hashed in all of the L functions. The points that lie in the same bucket as the query point are extracted from all of the functions and they form a candidate list. The exact near neighbors of q are then selected by exhaustive search over the candidate list.

The probability of successfully finding each one of the near neighbors for a set of parameters K and L (which we call $P_{success}$) is computable as follows. To get a data point p that is within distance r_1 from the query point q, at least one of the L functions must return that point. The probability that each one of the functions misses p is smaller than $1 - p_1^K$ and the probability that all the tables miss

p is smaller than $(1-p_1^K)^L$. Therefore, the success probability is:

$$P_{success} \geq 1-(1-p_1^K)^L \tag{2.5}$$

The time complexity of LSH can be decomposed into two steps:

1. *Query Hashing.* The complexity of this step is independent of dataset size and usually only a function of d, k and L.

2. *Searching the cadidate list* takes time proportional to the number of retrieved points. Assuming that the list contains $sel \times N$ where N is the number of dataset points, the search runs in $O(\frac{sel}{N}Ld)$.

The selectivity and query hashing times are presented for the LSH families in their respective sections.

For large datasets, the search step dominates because the size of the candidate list is dependent on the dataset size but the query hashing time is independent. For smaller datasets, the query hashing time becomes important. We will run our experiments on two datasets to see the effect of dataset size on different LSH functions.

While locality sensitivity is defined for any space and distance metric, because most similarity searches rely on euclidean distance, function families have been studied mainly for euclidean spaces. In [16] a function family is introduced for Manhattan distance. In 2004, Datar *et al.* [14] developed LSH families based on p-stable distributions. They also introduced E2LSH, a family of functions for euclidean distance which is of interest for us. We will discuss their contributions in more detail in the subsequent section.

Random Projections

This class of functions are used to estimate the cosine of angle between of points. Each function h is defined by a randomly rotated plane that goes through the origin. The plane can be represented by its normal vector n. Then:

$$h(v) = sign(v.n)$$

As we will see later, this family is similar to E2LSH but more costly.

Spherical LSH

Authors of [31] have developed Spherical LSH to solve the NNS problem when all the data points lie on the surface of a hyper sphere. In contrast to other LSH methods that try to partition the entire R^d space, spherical LSH tries to partition the surface of a $(n-1)$-sphere [2].

Spherical LSH uses randomly rotated regular polytopes to partition the surface of the hypersphere. There are only three types of regular polytopes in high dimensional spaces (when $d \geq 5$):

[2] an n-sphere, ofter written as S^n is a hypersphere embedded in R^{n+1}. S^n has an n dimensional surface.

- Simple: has $d + 1$ vertices and is analogous to tetrahedron.
- Orthoplex: has $2d$ vertices and is analogous to octahedron. Vertex coordinates are all permutations of $(\pm 1, 0, ..., 0)$.
- Hypercube: has 2^d vertices and is analogous to cube. Vertex coordinates are $\frac{1}{\sqrt{d}}(\pm 1, ..., \pm 1)$

It is possible to find the nearest vertex to a data point in O(d) for Simple and Orthoplex polytopes. For the hypercube, the search takes $O(d^2)$.

At first glance, this family seems to be suitable for our case where data points are normalized vectors. However, note that components of each SIFT feature only take positive values. That means all our points are in the $\frac{1}{2^d}$th of the unit sphere in the original coordinate system. Hence, the size of the buckets is too large in comparison to the dense distribution of our data points and even after rotation of the polytopes, they still reside only in a few buckets. We are interested in LSH families that map the points nearly uniformly in the bins.

E2LSH: LSH for Euclidean Distance

In E2LSH [14], each d-dimensional input point is projected onto K vectors $(a_i)_{1 \leq i \leq K}$ with random directions and unit length. The projections are then randomly shifted and discretized into bins of equal width. The hash value of an input vector v is computed by:

$$h_i(v) = \lfloor \frac{a_i.v + b_i}{W} \rfloor \text{ mod } N$$

$a_i.v$ is the length of projection of v onto a_i and b_is are uniformly chosen from $[0, W)$. W is the bin width parameter and should be chosen according to data. The discretized projection length is further mapped into $[0, N - 1]$ because of space considerations.

Each locality sensitive function $g_i \in \mathcal{G}$ is composed of K random projections to increase the discriminative power as described in Section 2.3.2:

$$g_j = (h_{j,1}, h_{j,2}, ..., h_{j,K})$$

where $h_{j,i}$s are selected randomly from $\mathcal{H}$. The rest of the algorithm works similar to basic LSH scheme described in Section 2.3.2.

Having L functions from $\mathcal{G}$, the query hashing time of E2LSH is $O(LKd)$ which corresponds to k dot products for every projection. There are $L \times K$ projections in total.

2.3.3 Improvements on LSH

After the original LSH algorithm was introduced in 1999, several strategies have been proposed to increase the quality of search in LSH. They either try to use space more efficiently or retrieve better candidate points. Note that in original LSH, we have to create L functions each of which takes space proportional to N (the size of our dataset). In literature it is normal for L to be as high as a few hundreds. It means that the tables can take up as much space as the data itself and space efficiency is a critical aspect for LSH algorithms.

We will discuss three major strategies that have been proposed for saving space along with mentioning some other strategies that are not of interest to us.

IE2LSH: Improved E2LSH

Computing the hash values constitutes a major proportion of E2LSH's query time. By reducing the number of projections (weak functions), it is possible to reduce the query hashing time. Indyk *et al.* [14] create g functions so that they reuse the weaker hash functions. We refer to their scheme as improved E2LSH (IE2LSH).

In their method, each function $g \in \mathcal{G}$ is made up of two smaller functions u_a and u_b. u functions are concatenation of $\frac{K}{2}$ weak functions.

$$u_a = (h_{a,1}, h_{a,2}, ..., h_{a,\frac{K}{2}}) \tag{2.6}$$

If m instances of u functions are created, it is possible to create $\binom{m}{2} = \frac{m(m-1)}{2}$ instances of g functions. It is possible to hash a query vector in all the g functions by computing only $\frac{mK}{2}$ projections. By using this improvement, the success probability of the LSH algorithm can no longer be computed according to Equation 2.5 because the hash functions are not independent. The new success probability is:

$$1 - (1 - p_1^{\frac{K}{2}})^m - mp_1^{\frac{K}{2}}(1 - p_1^{\frac{K}{2}})^{m-1} \tag{2.7}$$

Indyk *et al.* show that it is possible to set the LSH parameters with this improvement so that the query time is $O(dKm) = O(dK\sqrt{L})$. The number of hash tables will be higher than basic E2LSH scheme and the required space increases, but L will still be in the same order.

Entropy-Based LSH

In Entropy-based LSH (EB-LSH) [27], we create the hash functions the normal way. At query time, instead of returning only the points that are in the same bucket as the query points, we return points from some of the close by buckets. This way, we can get more candidates from fewer number of LSH functions to save space.

The main question in EB-LSH is which buckets to choose as there are many (i.e. in E2-LSH, there are $3^d - 1$ buckets adjacent to each bucket). The best way to pick buckets would be to sort them based on probability that they contain near neighbors of the query point. However explicit computation of those probabilities is cumbersome. As a compromise, in EB-LSH random points are generated on a hypersphere centered on q and hashed into buckets. The points are extracted from those buckets. Basically, by generating random points, we are sampling the probability function we avoided to compute explicitly. The distribution of the random points in nearby buckets is proportional to the probability of those buckets containing near neighbors of q.

While EB-LSH can save space by a factor of 3-8, it increases the query hashing time. Instead of hashing only the query point, one has to generate t rotation vectors ($O(d^3)$) and hash t more points ($O(tLkd)$). It is however possible to generate the rotation matrices offline. This approach is more suitable for large datasets where searching the candidate list dominates the query time.

Multi-Probe LSH

In Multi-Probe LSH (MP-LSH) [22], we extract points from multiple buckets that are adjacent to the query point bucket. The buckets are first sorted by their probability of containing near neighbors of query point and then they are checked in sequence until a specified number of candidate points are retrieved. The probability of containing a near neighbor of q, p, can be computed for different buckets based on the distance of q from the boundaries of its bucket. MP-LSH can achieve the same success probability as the basic E2LSH scheme by using only 15 percent of the memory that E2LSH uses.

Query Adaptive LSH

In Query Adaptive LSH (QA-LSH) [19], instead of creating L tables and looking up the query vector in all of them, we create more tables and then retrieve the candidates from the tables that are more likely to contain the near neighbors of the query vector. QA-LSH can reduce the query time by fetching candidates only from good bins for each query vector. But on the downside, the hashing time is higher compared to simple LSH. The space requirement is also higher than basic E2LSH strategy.

2.3.4 Discussion

We have to choose the LSH family and the specific variant of the algorithm that best suits our application. Euclidean distance is the distance measure that is being widely used for comparing SIFT features [20] and therefore E2LSH is a natural choice. E2LSH is also the most widely used LSH family function and the improvements have been made on this variant [27][19][22]. Therefore, we choose Euclidean LSH as our function family.

The problems that usually arise in practice when using LSH are memory shortage and parameter selection. We should be able to run our algorithms on ordinary computers that have a few gigabytes of RAM and in real time. The space requirement of basic E2LSH scheme is $O(LN)$ and L can become quite large according to literature (up to a few hundred). The constant for space requirement is 20 bytes in the naive implementation and in [14] the constant is reduced to 12 bytes. Even with the improved version, the space requirement for a 1M dataset of points will be around 3 gigabytes. Query Adaptive LSH creates more tables compared to basic E2LSH and storing them is infeasible for us. Entropy Based LSH and multi-probe LSH try to save space by creating fewer hash tables and are more suitable for us in this regard. Both QA-LSH and MP-LSH have a query time that is comparable to that of basic E2LSH. In [22], the authors run all three versions of LSH on a dataset of 1.3 million 64-dimensional data. The size of the dataset is of interest to us because we will be having roughly the same number of features in our experiments. They report the same query time for multi-probe LSH and basic E2LSH and a slowdown by a factor of 1.5 for entropy based LSH.

We work with basic E2LSH in this thesis. With basic E2LSH, we only study the cases where it is possible to fit the LSH tables in the main memory. If the program starts to use virtual memory, the performance of E2LSH will get worse by some orders of magnitude [2]. If one is able to tune the parameters of Multiprobe LSH, it will reduce the memory requirements by one or two orders of magnitude and might be more suitable.

2.3.5 E2LSH Parameter Setting

In [14], the authors propose a method to set the parameters of the basic E2LSH scheme. They ignore parameter W and mention that K and W have the same effect on the performance. If we decrease W, we can get the same performance by selecting a smaller K and vise versa. However parameter W should be selected with care in practice. In their method, W is picked in the beginning and L is expressed as a function of K and the success probability of the algorithm. The only parameter that remains is K and by empirically testing K values, it is possible to find optimal parameters.

In [2], the same authors propose a similar method for selecting parameters for their improved verison of E2LSH. Again, parameter W is pre-selected and the expected query time must be evaluated on data. This step can become time consuming if the dataset is large. If the data is not available at the time of creating the LSH tables, it is possible to use sample data from similar datasets.

To the best of our knowledge, no method for setting the parameters of QA-LSH, MP-LSH or EB-LSH has been proposed. Each of these improvements uses the parameters of basic E2LSH and has some extra parameters. If one is to set the parameters of these improved versions, he should have a method to estimate the running time and success probability of them. In this thesis we will try to find the optimal parameters of the basic E2LSH scheme without empirical evaluation of parameters on the data. We will show that some statistics of the data (in our case distance distribution of the data (Section 3.3)) are sufficient for optimizing the parameters due to the fact that the performance of LSH scheme is inherently dependent on the distribution of the data.

Chapter 3

Optimal Selection of LSH Parameters

In this chapter we describe our approach for setting the parameters of LSH algorithm. Datar *et al.* [14] provide a method for tuning the parameters for a specific dataset so that the success rate of the algorithm is higher than a given threshold. However, they do not explain how to select bin widths W precisely. They only mention that the performance will be stable beyond a small value (compared to distance of the points in the dataset) of W. Their method also requires testing different k values to find the optimal parameter values and one has to test each setting on sample data.

We would like to be able to adjust the parameters of the LSH tables prior to our experiments. It is desirable to set the parameters independent of the specific data that we see in the experiment. LSH has three parameters and parameter setting is one of the difficulties of effectively using this approach. In this section we present a strategy for setting the parameters in a way that the running time and space requirements of the algorithm meets our bounds. This strategy is useful when we need a *hard* real-time system: the running time is not flexible at all and we must decide the loop closures within the given time frame.

In Section 3.1 we derive an equation for the running time of E2LSH. In Section 3.2 we derive the equation for collision probability in a single hash function h using surface area of hypersphere caps. We continue by studying distance distributions in Section 3.3. We show how we can characterize the *homogeneity* of a dataset in this section. In Section 3.4, we use the distance distribution of a dataset to estimate the expected running time of E2LSH. Finally, we show the relation between a parameter setting and time and recall of E2LSH and conclude the chapter.

3.1 Calculating the Running Time

Here we aim to set the parameters so that the query time for a given vector is bounded. The query time in E2LSH can be split into two parts: the hashing time (T_h) and the search time (T_s) (see

Section 2.3). Our goal is to set the parameters so that:

$$T_h + T_s < \tau \tag{3.1}$$

where τ is the time limit on query time. T_h and T_s are related to E2LSH parameters L, K and W and dimensionality of the vectors, d, by the following relations:

$$T_h = O(LKd) \tag{3.2}$$

$$T_s = O(n_u d) \tag{3.3}$$

n_u is the expected number of unique candidate vectors retrieved from the LSH tables. The vectors retrieved from one table are guaranteed to be unique; however the vectors that are retrieved from different tables may contain duplicates. We compute the expected number of duplicate vectors and set the parameters to impose the bound on it. Because of our matching criterion, the number of the candidates we require is very low. We only check the candidate vectors until there is a large enough gap between the distances of the candidate vectors and the query vector (see Section 4.4.2).

3.2 Computation of Collision Probability for the Weak Hash Functions

Each weak hash function in E2LSH has three parameters: the random vector a, the bin width parameter W and the random shift $b \in [0, W)$. Suppose we have a single hash function h. We want to compute the probability of collision for two vectors v_1, v_2 in d dimensions, selected randomly at distance r from each other. Remember that the hash value for a vector is simply the index number of the bin to which it is projected. Without loss of generality, we assume v_1 is at the origin and we replace v_2 with $v_2 - v_1$. Now v_2 has a uniform distribution on a (d-1)-sphere of radius r around the origin. The probability of collision is the ratio of the surface of the sphere that is in the same bin as the origin (S^d_{col}) to the surface of the sphere (S^d):

$$P(h(v_1) = h(v_2)) = \frac{S^d_{col}}{S^d} \tag{3.4}$$

S^d_{col} is related to the surface area of the caps that are formed on the sides of the collision bin:

$$S^d_{col} = 1 - \frac{S^d_{cap}(u_1) + S^d_{cap}(u_2)}{S^d} \tag{3.5}$$

where u_1 and u_2 are distance of the origin to the edges of the bin that contains it. We use this formulation because the formula for computing the surface area of a sphere cap is available and we will be able to use it by Equation 3.5.

Figure 3.1 shows sample surfaces in 2 dimensions for demonstration. S^d_{col} corresponds to the length of the red parts of the circumference of the circle and S^d is the total circumference of the circle. $S^d_{cap}(u_1)$ and $S^d_{cap}(u_2)$ correspond to the black parts of the circle that are formed on both sides of the red curves.

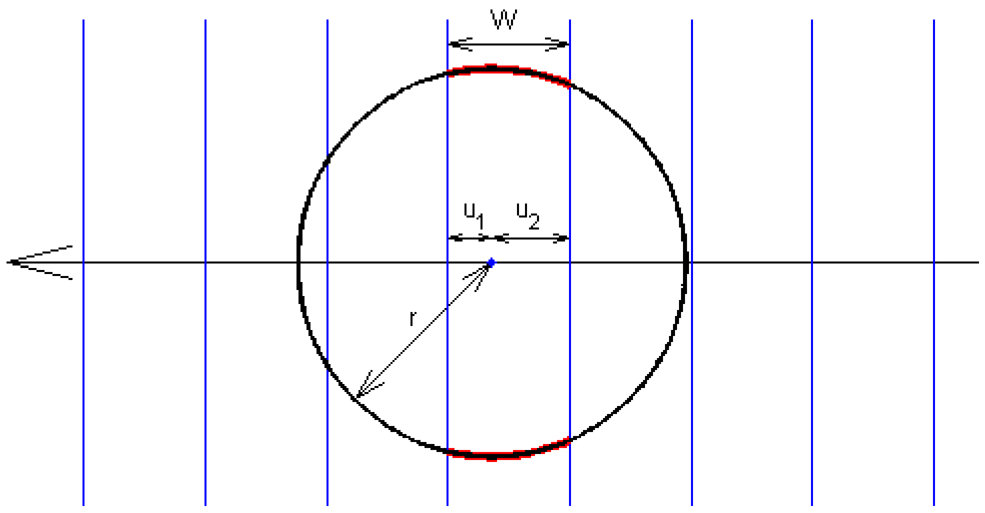


Figure 3.1: Demonstration of computation of probability of collision for a single hash function. The vector represents the function and the blue lines mark the margins of the bins. The ratio of the length of the red parts to the circumference of the circle is the probability of collision.

The total surface of a d-sphere of radius r is computable from:

$$S^d = (d+1)C_{d+1}r^d \tag{3.6}$$

where C_d is given by:

$$C_d = \frac{\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2}+1)} \tag{3.7}$$

It is possible to compute the cap area with the help of the incomplete beta function:

$$S^d_{cap}(u) = \frac{\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2})} r^{d-1} I_{\frac{(r^2-u^2)}{r^2}}(\frac{d-1}{2}, \frac{1}{2}) \tag{3.8}$$

The incomplete beta function $I_a(b,c)$ is defined as:

$$I_x(a,b) = \frac{\int_0^x t^{a-1}(1-t)^{b-1}dt}{\int_0^1 t^{a-1}(1-t)^{b-1}dt} \tag{3.9}$$

We use Equation 3.8 to compute the surface of the sphere caps. Note however that the incomplete beta function does not have a closed form and its value is computed by numerical methods. Figure 3.3 shows the surface ratio of caps of different sizes in 16, 64 and 128 dimensions.

In [15] and [1] a bound of the surface has been used instead of the exact surface areas. According to [15] the surface area of a cap of a d-sphere that starts at distance u from the center of the d-sphere

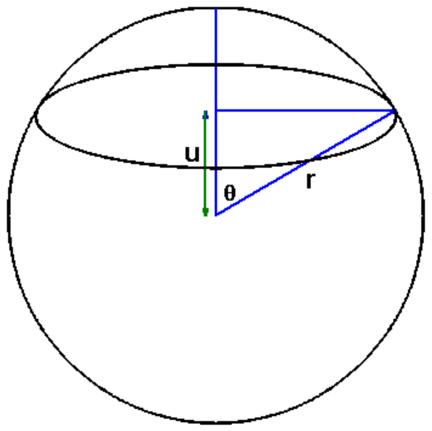


Figure 3.2: Sphere cap at distance u from sphere center. For demonstration of Equation 3.8.

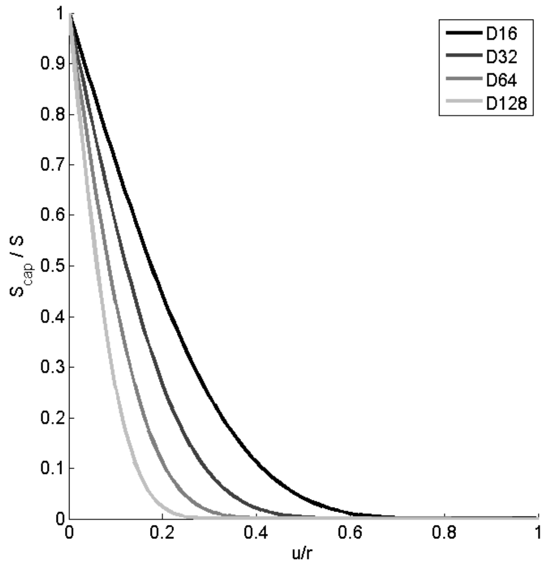


Figure 3.3: Ratio of cap surface area to sphere surface as a function of u in different dimensions. As the dimensionality increases, the surface area of caps decreases more sharply as the base of the cap moves away from the center of the hyper-sphere

(Figure 3.2) is bounded by:

$$S^d_{cap}(u) \leq \frac{1}{2}\big(1-(\frac{u}{r})^2\big)^{\frac{d}{2}} S^d$$
$$\Rightarrow \frac{S^d_{cap}(u)}{S^d} \leq \frac{1}{2}\big(1-(\frac{u}{r})^2\big)^{\frac{d}{2}} \tag{3.10}$$

We will use this bound for setting the parameters in addition to exact computation. The bound is not tight enough and the parameters that are selected by using them do not perform as well as exact computation. To get a feel of how the bound relates to actual cap surface, in Figure 3.4 we plot the

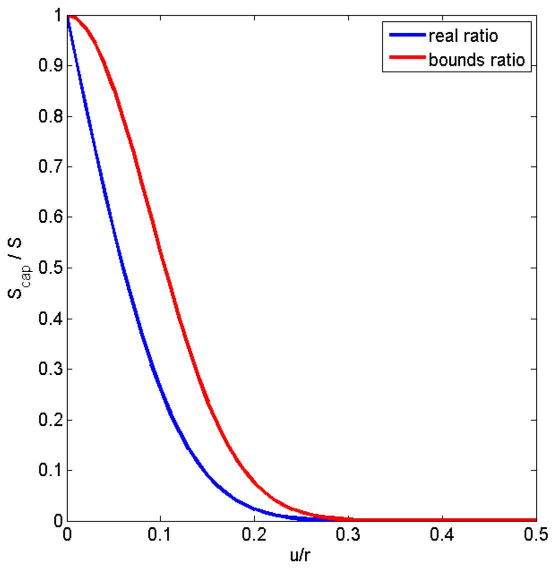


Figure 3.4: Actual ratio of cap surface vs. ratio computed from bounds. Both ratios get very close to 0 quickly.

actual surface area ratio along with bounds of the ratios of 127-sphere caps of different sizes.

In Equation 3.4, we have computed probability of collision for a given h. If the function is selected randomly from the family $\mathcal{H}$, we have to average the probability over all possible h. Note however that because of the symmetry of sphere, it does not matter what the random vector is within h. We have to integrate only over the shift values:

$$\begin{aligned} P_{col}^{h}(r) &= \int_0^W P(h_b(v_1) = h_b(v_2))db && (3.11)\\ &= \int_0^W \frac{S_{col}^d(b)}{S^d}db && (3.12)\\ &= 1 - \int_0^W \frac{S_{cap}^d(\min(r,b)) + S_{cap}^d(\min(r, W-b))}{S^d}db && (3.13)\end{aligned}$$

We are unable to get a closed form formula for this equation because the term inside the integral itself does not have a closed form. So we do this computation by numerical methods. We use the collision probabilities for computing the parameters and the efficiency of the computation is not a concern for us. Figure 3.5 shows the probability of collision as a function of $\frac{W}{r}$, computed from Equation 3.11 using exact area of sphere caps and the bounds of Equation 3.10. Parameter W depends on the distance of points that we want to retrieve. If all the points in the dataset are scaled, we should scale W by the same factor to get the same expected number of candidates. P_{col} is a function of $\frac{w}{r}$ or the distance between the points. Therefore, the probability of collision can be unstable for different query features if the distance of the data points to them has high variations. In the next section, we will introduce distance distributions and define certain metrics on them to help us compute selectivity of hash functions.

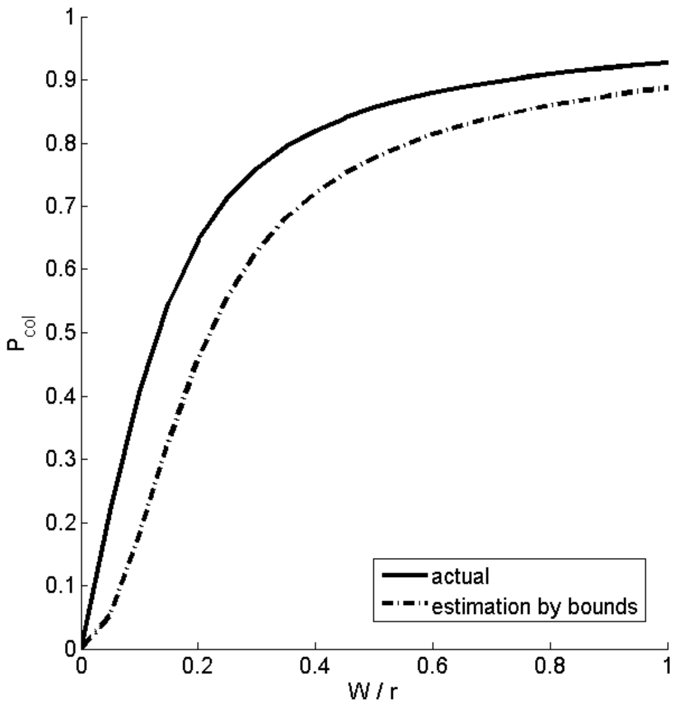


Figure 3.5: Probability of collision of two points in a randomly selected hash function h as a function of $\frac{W}{r}$. Only small values of W are of interest to us. Values larger than 1 definitely lead to infeasible running times.

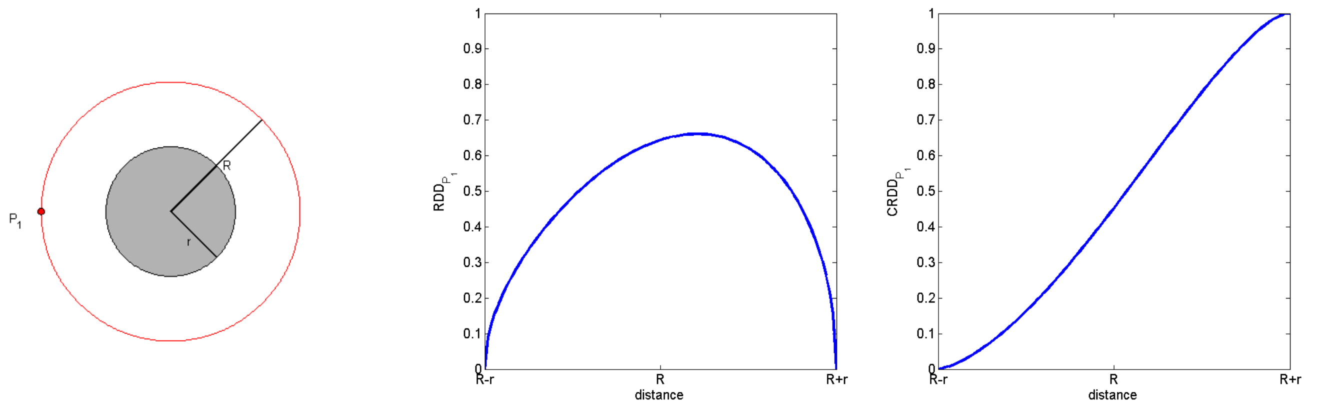
3.3 Distance Distributions

Distance distributions have been previously used in approximate nearest neighbor search. In [11] the authors define a concept of homogeneity for distance distributions and use it to develop and analyze a cost model for NN queries in metric spaces using M-trees. Ciaccia *et al.* [10] use distance distributions in cost analysis of their PAC-NN (Probably Approximately Correct) algorithm. We will define overall and relative distance distributions, discrepancy of distance distributions and a measure of homogeneity of viewpoints here. For additional information on these topics refer to [11].

Distance distributions are useful for analysis of NN query problems. The distance distributions are directly derivable from data distributions. Even if the query vectors are biased, that is they come from a different distribution than the data distribution, the distance distributions can be used.

Consider a bounded random metric (BRM) space, $M = (U, d, d^+, S)$ where U and d are the domain of the data and the distance metric as any metric space, d^+ is a finite bound on the distance of the data points and S is the data distribution over U. The overall cumulative distance distribution of M is defined as:

$$CDD(x) = Pr\{d(P_1, P_2) \leq x\} = \int_0^x DD(x)dx \qquad (3.14)$$



(a) A BRM space (b) Relative distance distribution of P_1 (c) Cumulative relative distance distribution of P_1

Figure 3.6: Example demonstrating relative distance distributions. The gray disk in (a) shows the distribution of points in space. (b) and (c) show DD_{P_1} and CDD_{P_1} respectively. The RDDs for all the points on the red circle are the same as DD_{P_1}.

P_1 and P_2[1] are two independent S-distributed random points over U. The relative cumulative distance distribution for $P_i \in U$, CDD_{P_i}, is given by:

$$CDD_{P_i}(x) = Pr\{d(P_i, P_2) \leq x\} = \int_0^x DD_{P_i}(x)dx \tag{3.15}$$

DD_{P_i} can be viewed as P_i's view point of S. We demonstrate the distance distribution concept with an example:

Example 1 Consider the BRM space $([-R, R]^2, L_2, 2R\sqrt{2}, S)$ as depicted in Figure 3.6a. The points are uniformly distributed on a disk of radius $r = \frac{R}{2}$ centered on the origin (the gray disk corresponds to S). For a point P_1 at distance R from the origin, the relative distance distribution and cumulative distance distribution are shown in Figure 3.6b and Figure 3.6c respectively. The RDD and CRDD for any other point at distance R from the origin (the red circle) would be the same as P_1.

Points in our space can have different view points and a notion of similarity between the view points is required. The normalized absolute difference of two view points, referred to as discrepancy, is the measure used in [11].

Definition 1 *The discrepancy of two RDDs, DD_{P_i} and DD_{P_j} is defined as:*

$$\delta(DD_{P_i}, DD_{P_j}) = \frac{1}{d^+} \int_0^{d^+} ||DD_{P_i}(x) - DD_{P_j}(x)||dx \tag{3.16}$$

The discrepancy is a real number in [0,1]. A discrepancy value of 1 means that the viewpoints are very dissimilar and a discrepancy of 0 means that the view points are the same. Note however that if $DD_{P_i} = DD_{P_j}$, it does not mean that $P_i = P_j$ (see Example 1).

[1] P_1 and P_2 are equivalent to v_1 and v_2 in the previous sections. We change notations here to be consistent with the reference.

Now for two points P_1 and P_2 selected uniformly at random from U, $\Delta = \delta(DD_{P_1}, DD_{P_2})$ is a random variable and $G_\Delta(y) = Pr\{\Delta \leq y\}$ is the probability that the discrepancy of the two RDDs is not larger than y. A higher value of G indicates that the two RDDs are more likely to behave the same. If $G_\Delta(y) \simeq 1$ we say that the BRM space M is homogeneous with respects to distribution S. By defining an index of homogeneity we can formulate this:

Definition 2 The index of "Homogeneity of Viewpoints" of a BRM space M is defined as:

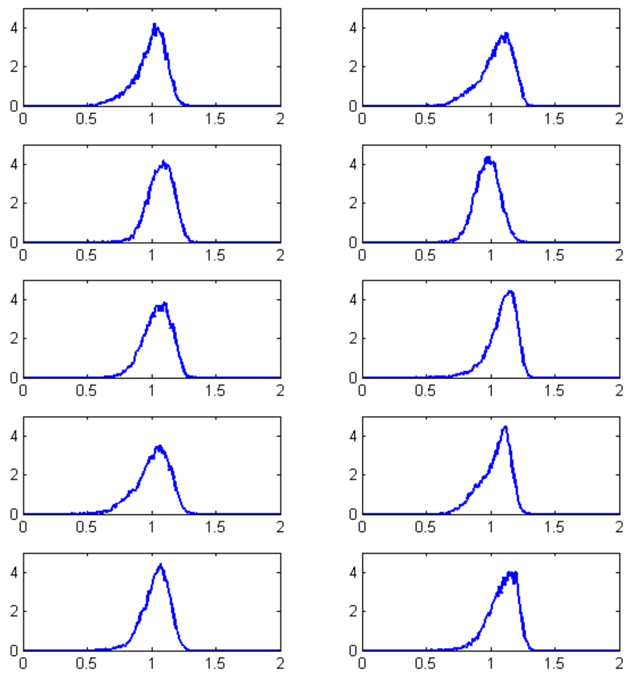
$$HV(M) = \int_0^1 G_\Delta(y)dy = 1 - E[\Delta] \tag{3.17}$$

This homogeneity index, describes how likely random points from a space are to have similar view points. A value close to 1 means that the space is fairly homogeneous. This index can be used as a general index if the distribution of the query points is unknown. However, if the distribution of the query points is different from uniform distribution, the homogeneity index might be inaccurate. Consider the case in Figure 3.6. The query points lie on the boundary of a circle. All the points that lie in any circle with the same center, will have the same viewpoint of M. If our query distribution is such a distribution, then the homogeneity index should be 1. $HV(M)$ is however not one because the points that lie on circles with different radii will have some discrepancy.

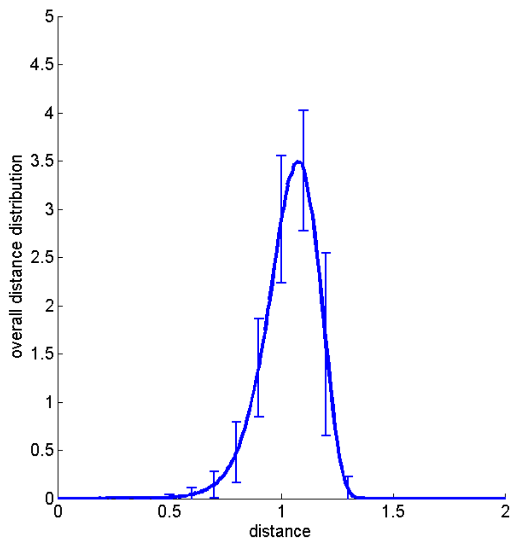
It is possible to extend the definition of the homogeneity index to cases where a query distribution is present. Assume that the query points come from distribution T over U. It suffices to substitute T instead of uniform distribution in definition of Δ. The rest of the equations remain unchanged. We denote the homogeneity of a BRM space with respect to a query distribution by $HV^T(M)$.

Exact computation of the homogeneity requires the availability of the data and query distributions and computing the integration over arbitrary hyper-volumes (positive parts of a hyper-sphere in our case) might be cumbersome. Because we can only obtain empirical distributions with SIFT features, which are 128-dimensional normalized vectors, we compute the distance distributions empirically. We used 100K SIFT features as our data and 100 features as query vectors. Figure 3.7a shows sample local distance histograms for 10 query SIFT vectors. Figure 3.7b shows the overall distance histogram for SIFT features. The distance distributions show that the features are concentrated at around distance 1 of other features. Only few features are closer than 0.5 to any SIFT feature.

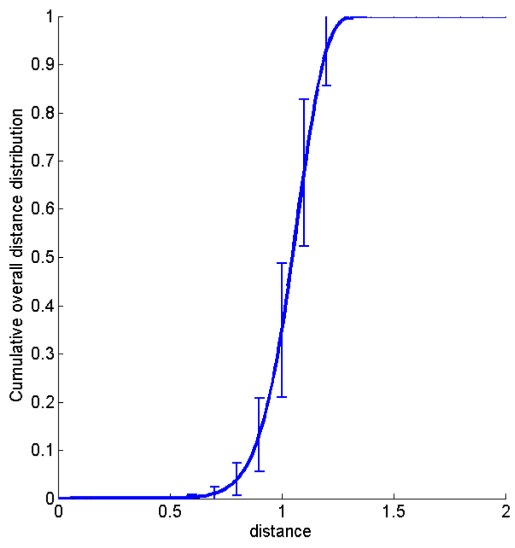
In order to be able to use the overall distance distribution of SIFT features, we should make sure that the distribution does not vary greatly in different datasets. To do that, we computed the distance distributions for our other datasets. Figure 3.8 shows the overall distance distributions for all the datasets. As the graphs show, the distance distribution of SIFT features is stable over different datasets. Table 3.1 shows the pairwise discrepancy of overall distance distributions of the datasets. These low discrepancy values show that the distance distribution is similar in all the SIFT datasets. This allows us to talk about the distance distribution of SIFT features, without referring to any specific dataset.



(a) Sample relative distance distributions of SIFT features



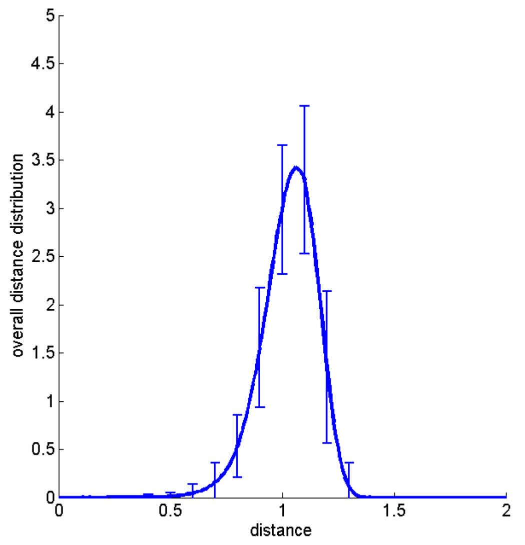
(b) Overall distance distribution of SIFT features



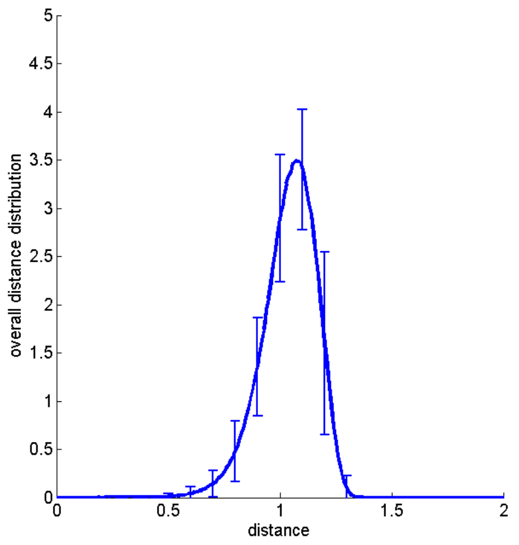
(c) Cumulative overall distance distribution of SIFT features

Figure 3.7: Distance distributions of SIFT features. Figure (a) shows 10 sample SIFT RDDs to show the amount of variations in them. Figures (b) and (c) show the DD and CDD of SIFT features with their standard deviations.

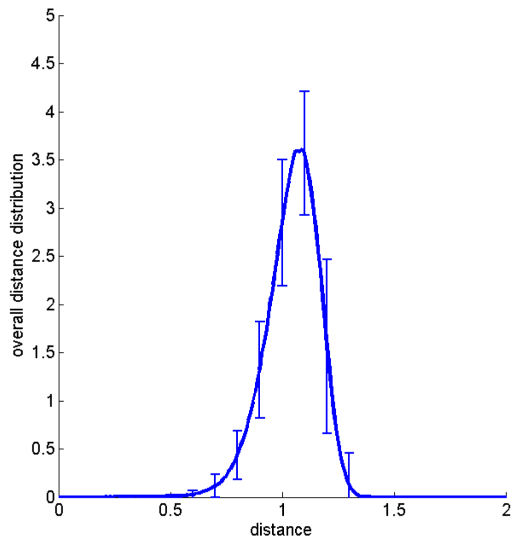
In addition to distance distribution of SIFT features, we compute another distance distribution from a dataset of points that is uniformly distributed over the positive coordinates of a unit sphere



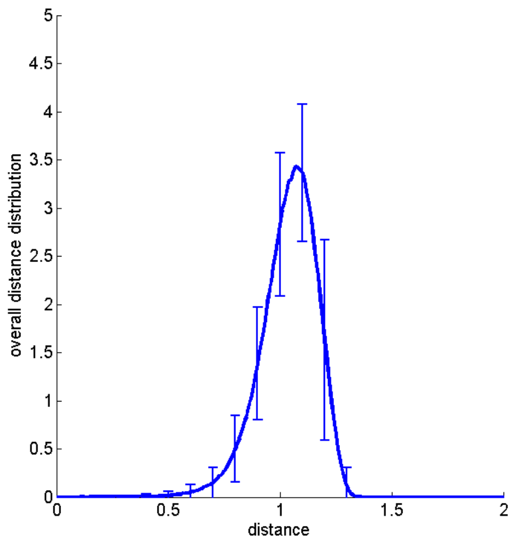
(a) Distance distribution of panoramic GooglePits dataset



(b) Distance distribution of calibrated GooglePits dataset



(c) Distance distribution of Auxiliary dataset

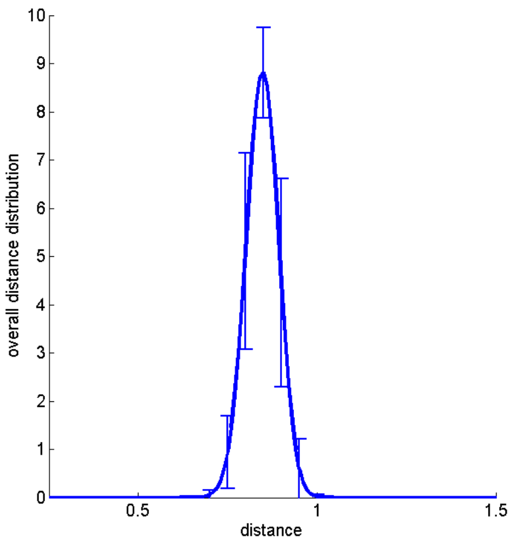


(d) Distance distribution of City Center dataset

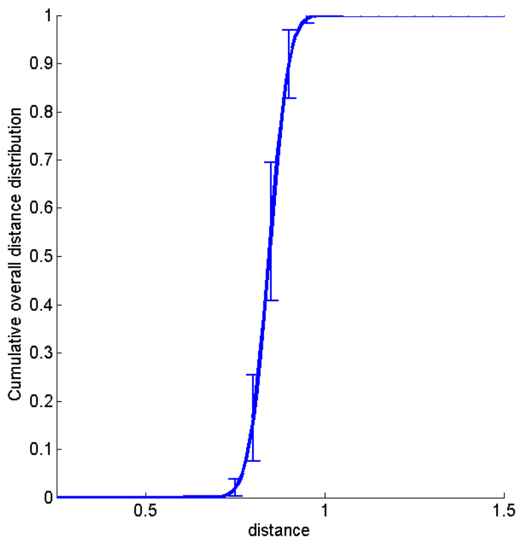
Figure 3.8: The overall distance distribution of different SIFT datasets. The distance distribution of SIFT features is fairly stable in different datasets (see Table 3.1).

in 128 dimensions. This is the simplest distribution one can assume for a set of data points. The homogeneity index for both SIFT features and our random dataset is shown in Table 3.2. These high values show that the local distance distributions for individual query vectors will be fairly the same in both cases.

From the distance distributions $E[\Delta]$ can be computed by first obtaining pairwise discrepancies between local distance histograms and averaging them.



(a) Overall distance distribution of Random vectors



(b) Cumulative overall distance distribution of Random vectors

Figure 3.9: Distance distributions of random vectors. Figures (a) and (b) show the DD and CDD of random generated vectors with their standard deviations.

Table 3.1: Discrepancy of overall distance distributions in different SIFT datasets

Dataset	GPP	GPS	CC	Aux
GPP	0	0.0087	0.0048	0.0049
GPS	0	0	0.0042	0.0053
CC	0	0	0	0.0037

Table 3.2: Homogeneity Index of Random and sample SIFT datasets

Dataset	$HV^S(M)$
random	0.981
SIFTs	0.951

3.4 Expected Selectivity of Hash Functions

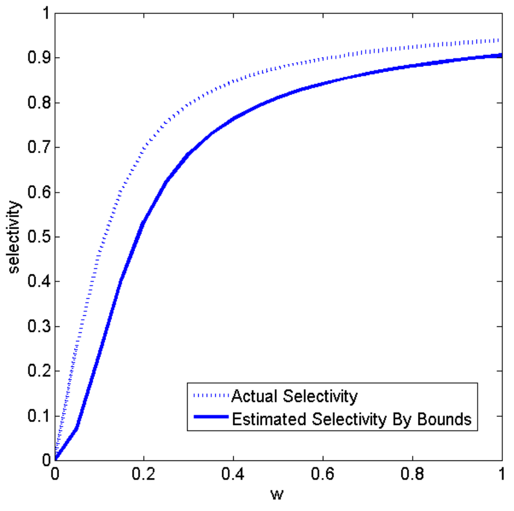
The probability of collision for each weak function is independent given the distance of the points:

$$P[h_1(v_1) = h_1(v_2), h_2(v_1) = h_2(v_2)] \tag{3.18}$$
$$= P[h_1(v_1) = h_1(v_2)|d(v_1, v_2)]P[h_2(v_1) = h_2(v_2)|d(v_1, v_2)]$$

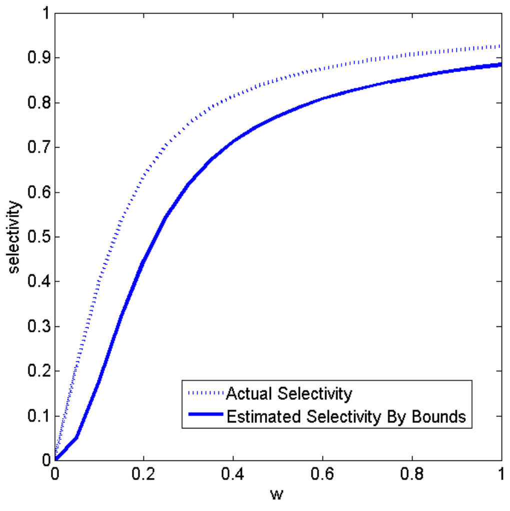
Using this observation, for a single hash function h, the expected value of selectivity, $E[S_h]$ is computable from:

$$E[S_h] = \int_{x=0}^{d+} DD(x)P^h_{col}(x)dx \tag{3.19}$$

Figure 3.10 plots the expected selectivity of a single hash function h



(a) Selectivity of functions for randomly generated data



(b) Selectivity of functions for sample SIFT data

Figure 3.10: Expected selectivity of weak hash functions h on uniformly distributed points and SIFT features as a function of W. As W increases, $E[S_h]$ approaches 1. We are interested in small W values below 0.3 which is the distance at which the SIFT features match. The selectivity for SIFT features is very close to that of the randomly generated points.

The relation between the collision probability of two points in a generalized hash function g is related to that of of h by Equation 2.5. For L instances of generalized functions g, the selectivity is given by:

$$E[S_{L,K}] = \int_{x=0}^{d+} DD(x)\Big(1 - \big(1 - (P_{col}^{h}(x))^{K}\big)^{L}\Big)dx \tag{3.20}$$

To use this equation the pdf of distances of data points in the dataset ($DD(x)$) and the probability of collision of two points in a single weak hash function are required. It is also possible to compute selectivity of IE2LSH hash functions. All we need to do is to replace the collision probability of those improved functions in Equation 3.20. The equation for expected selectivity of an improved E2LSH structure becomes:

$$E[S_{m,K}] = \int_{x=0}^{d+} DD(x)\Big(1 - (1 - p^{\frac{k}{2}})^{m} - mp^{\frac{k}{2}}(1 - p^{\frac{k}{2}})^{m-1}\Big)dx \tag{3.21}$$

where p is $P_{col}^{h}(x)$.

Assuming that we know $E[S_{L,K}]$, to impose a time limit τ on the query time, the parameters must satisfy:

$$T_h + T_s < \tau \tag{3.22}$$

$$LKd + E[S_{L,K}]Nd < \tau \tag{3.23}$$

Parameter W does not appear in equation Equation 3.23 and is hidden in $E[S_{L,K}]$ term.

Note that we only have the expected number of candidate features but in each individual query, there may be many more or less candidates depending on the distribution of our data. Unless we make fundamental changes to E2LSH algorithm (like having buckets of different sizes), we cannot do anything about this variance in the number of candidates. To meet hard time constraints, one can look up the query vector in LSH tables until enough number of candidates are found and ignore the rest of the tables. We use this strategy in our experiments.

Once we have $E[S_{L,K}]$ we are able to compute the expected query processing time from Equation 3.23. There are three degrees of freedom while we have to satisfy two inequalities (time constraints and space constraints). It is logical to select the probability of successfully finding the true nearest neighbor (Equation 2.5) as a criterion for optimization. If we find the nearest neighbors more accurately, the performance of feature matching will also increase. To use Equation 2.5 one must know the probability of collision given the distance of two vectors and also the distance at which the feature points match.

3.5 Distance Threshold for SIFT Features

To get a threshold on the distance of SIFT features, we use a sample dataset of SIFT features extracted from Google Pits dataset. The features of images that are taken from the same places are matched by using the distance ratio technique (see Section 2.2.2). The matched features are then manually verified to ensure they actually come from the same object. We use this threshold to compute the success probability of our nearest neighbor search and optimize parameters of E2LSH. Also, we use the same threshold for matching features (see Section 4.4.2).

Figure 3.11 shows the distance distribution of matched and unmatched SIFT features. The black line shows the distance distribution of matched sift features and the blue line shows the distance distribution of unmatched features. These curves are normalized to have the same area and one should keep in mind that the number of unmatched feature pairs is larger than matched features by some orders of magnitude. If only a small proportion of unmatched features get accepted, say 1 percent, that would still be much more than the matching features. We should set the parameters of LSH so that only a few of unmatched features get retrieved. By setting the threshold to 0.5, we will get 91 percent of the matching features while we allow only 2 thousandth of the non-matching features to pass.

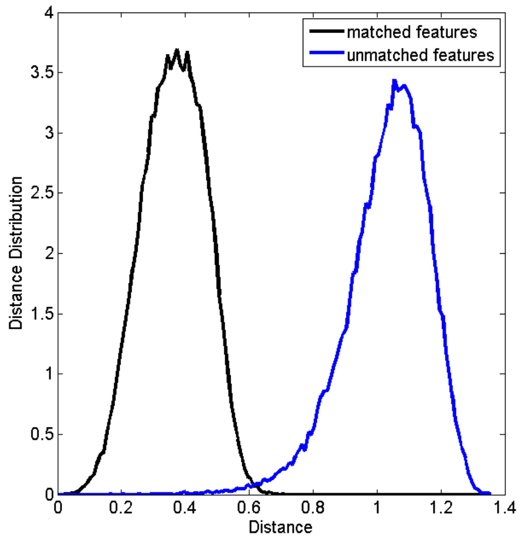


Figure 3.11: Overall distance distribution of matched and unmatched SIFT features. The little overlap between the plots shows the euclidean distance is a reasonable choice for matching SIFT features.

3.6 Solving for Optimal Parameters

E2LSH

By using the distance threshold from the previous subsection, we are able to use Equation 2.5 to optimize the parameters. What we want to do is to find L, K and W to maximize probability of success:

$$\begin{aligned} & P_{success} = 1 - (1 - p^K)^L \\ s.t \quad & c_1 LKd + c_2 E[S_{L,K}]Nd < \tau \\ s.t \quad & cLN < s \end{aligned}$$

where $p = P^h_{col}(r)$. The constants (c_1, c_2, c) depend on the implementation. One can estimate them by counting the number of instructions or empirically by running the hashing part and linear search part individually.

The objective function and the constraint equation are not linear and it is possible that there are many local optimums that satisfy the constraints. It might be possible to use Lagrange multipliers, general function optimization methods or other analytic methods to solve the problem. We use a simple approach to find the optimal parameter values. We discretize W and sweep the interesting area of K and W parameters. We try W values from 0.05 to 0.35 in 0.025 intervals and vary K from 1 to 40 for each W. For each W-K pair, we start trying L values from 1 upwards until the success probability of the parameters passes the threshold. At this point, if the running time and space requirement meet the constraints, we accept the parameters. Because for each parameter combination we have to spend less than a second, the whole process can be done in a few minutes.

Figure 3.12 shows the success probability, optimal number of hash tables and the running time as functions of K and W. For each W value, the success probability is low for small and large values of K. Success probability has a peak somewhere in moderate values of K. The low success rate on small values of K is due to high selectivity of hash functions in this region. The tables return most of the vectors as candidate vectors and most of the time is spent on linearly searching vectors that are selected nearly randomly. The performance is bound by the time limit in this region and most of the time is spend on the linear search of the candidate list. On the other side, for high values of K, the functions have very low selectivities. Most of the query time is spent on hashing the query vector and very few candidates get retrieved from the tables. The performance is bounded either by the space limit or query hashing time in this region of K. All the curves in the high K region converge to the same value which is the maximum number of tables for which hashing the query is feasible withing the time limits. The optimal success rate appears somewhere around the place that the algorithm takes up all the space. This is place where the performance is bound by both space and time limits.

As an example of how to select the best parameters, consider the curves in Figure 3.12a. The green curve has the highest peak, so the optimal value of W is 0.1. To be more precise, the optimal value of W is somewhere between 0.1 and 0.15 (between the green and blue curves). It is possible to find more precise values of W by checking Ws at smaller intervals or using binary search. After finding the optimal W, K and L that correspond to the peak of the curve should be selected. In the case of Figure 3.12, $K = 12$ and $L = 170$ are optimal.

IE2LSH

The equations for IE2LSH are similar to that of E2LSH. The goal is to maximize probability of success:

$$\begin{aligned} &P_{success} = 1 - (1 - p^{\frac{K}{2}})^m - mp^{\frac{K}{2}}(1 - p^{\frac{K}{2}})^{m-1} \\ &s.t \qquad c_1\frac{m(m-1)}{2}Kd + c_2E[S_{m,K}]Nd < \tau \\ &s.t \qquad cLN < s \end{aligned}$$

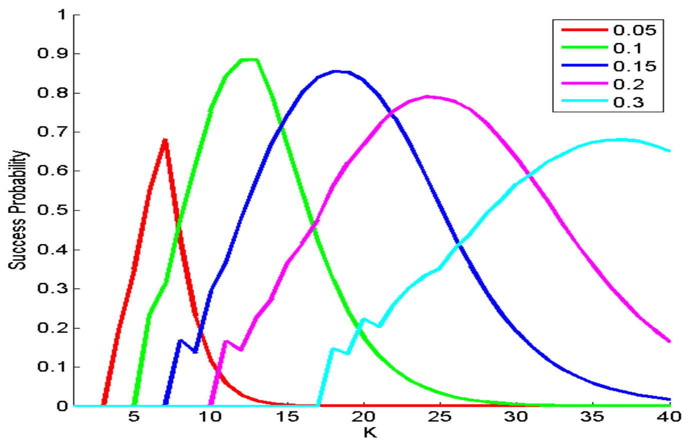
where $p = P^h_{col}(r)$. We sweep the parameter space similar to previous section. We try W values from 0.05 to 0.35 in 0.025 intervals. K has to be an even number in this case, so only the even values from 2 to 40 are tested. Figure 3.13 shows the success probabilities, running times and optimal m values for different values of W and K. The time and space limits we used are the same as previous section.

Similar to E2LSH, for each W value, the success probability has a peak somewhere in moderate values of K. The plots are less smooth compared to E2LSH because of the K intervals are coarser here. For small K values, most of the time is wasted in linearly searching candidates that are selected nearly randomly. On the high K values, most of the time is spent on hashing. However, in contrary to the case with E2LSH, the performance is bound by the space limit in this case. For all W values, the algorithms are using as many tables as they can (Figure 3.13c) but the running times converge to the same value which is the time needed to hash a query vector in the maximum number of tables. Again, the optimal performance is achieved when tables start to use all the space that is available to them.

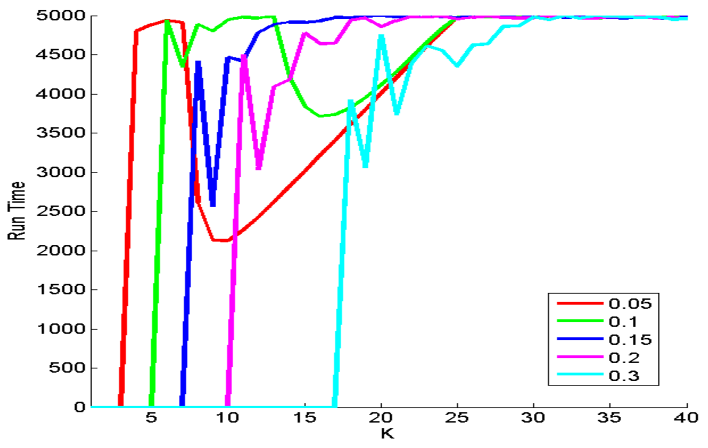
In the above cases, the performance of E2LSH scheme was superior compared to IE2LSH. Note however that if the space limit was higher or the time limit was tighter, then IE2LSH would have outperformed E2LSH.

3.7 Conclusions

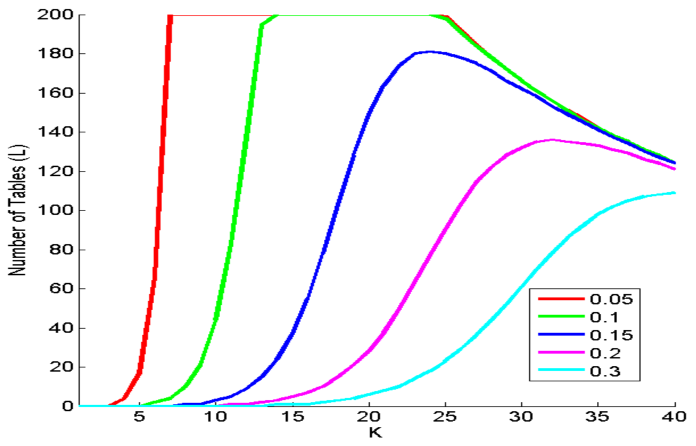
In this section we developed an algorithm to use LSH for finding similar features in real-time. Our main contribution was setting the parameters without testing them on real data by using the distance distribution of the data. We showed empirically that the distance distribution is stable for SIFT features. We also demonstrated how we can extend the distance ratio criterion for matching features in multiple images.



(a) Success probability as a function of W and K

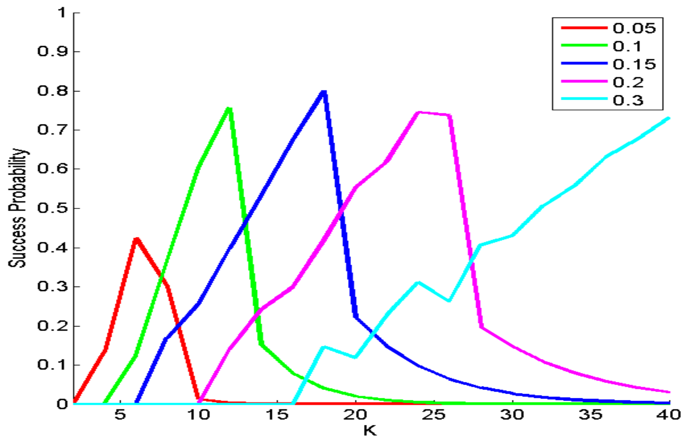


(b) Running time as a function of W and K

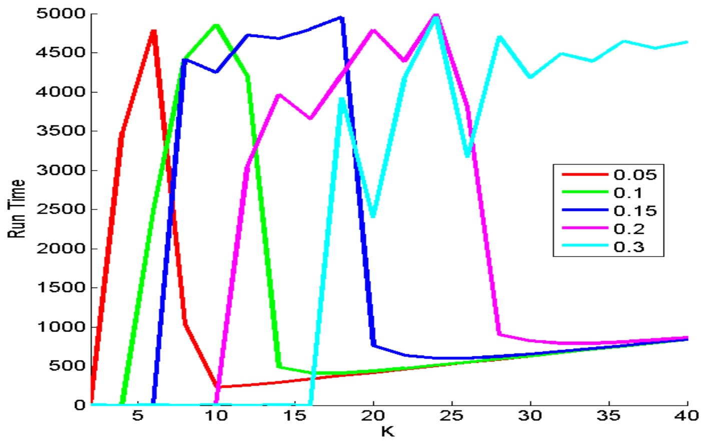


(c) Optimal L value as a function of W and K

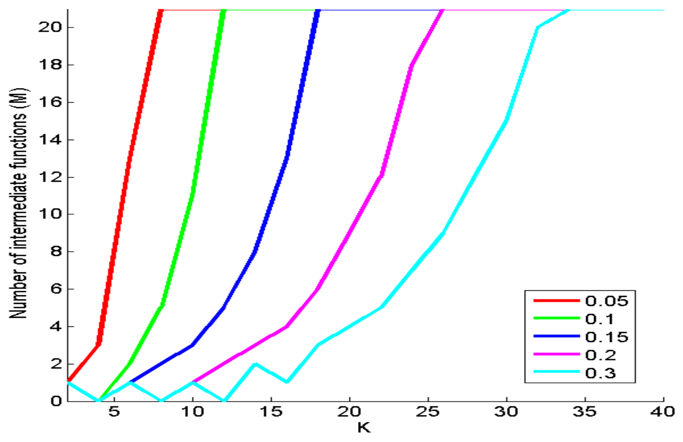
Figure 3.12: The (a) success probability, (b) running time and (c) L optimal value as functions of W and K for E2LSH. The time limit was set to 400K operations (3K vector dot products). The maximum number of tables was set to 200 (roughly 3GB).



(a) Success probability as a function of W and K



(b) Running time as a function of W and K



(c) Optimal L value as a function of W and K

Figure 3.13: The (a) success probability, (b) running time and (c) L optimal value as functions of W and K for IE2LSH. The time limit was set to 400K operations (3K vector dot products). The maximum number of tables was set to 200 (roughly 3GB) which implies that $m \leq 21$.

Chapter 4

Experiments

We run three sets of experiments. The first set involves verifying our selectivity prediction method. In these experiments, we try different parameter settings on our datasets and compare the empirical results with that of our analysis. In the second set of experiments, we evaluate our parameter selection method. We report the nearest neighbor search accuracy of E2LSH. In the third set of experiments, we use E2LSH scheme to find loop closing images in two urban datasets. We compare the results with that of the BOW approach. The empirical results show effectiveness of our method in real datasets.

4.1 Datasets

We use four datasets in our experiments. Three of the datasets contain SIFT features from urban images. One dataset contains randomly generated vectors. We use all datasets for our first two sets of experiments. In those experiments we evaluate our selectivity prediction and parameter selection methods. We run our SLAM experiments on two large scale urban datasets.

The first large scale SIFT dataset, called *city center* (referred to as CC in short), is compiled by [12]. The images are gathered by a robot by going twice through a loop of length 1 kilometer. The images are approximately 1.5 meters apart. The dataset contains 2474 non-calibrated images in total that are taken from left and right of robot's trajectory. In our experiments we used only one set of the images (the left view) because of memory limit considerations. Figure 4.1 shows sample images from this dataset.

The second large scale urban dataset is called *Google Pittsburgh* dataset. It contains an image sequence gathered by Google for Street View feature of Google Maps. The dataset contains around 12K panoramic images. For each location, there are four additional projections: left, right, front and back, obtained by unwarping the panoramic image. The side views are more suitable for our work because they are calibrated. This dataset is more challenging compared to the city center dataset. It has a longer trajectory and there are multiple loop closures for some locations. Again because of memory limit considerations, we use a subset of 2700 images from this dataset. We select one image

Figure 4.1: Sample images from the City Center dataset. The images are not calibrated.

Figure 4.2: Sample images from the GPS dataset. The images are not calibrated and for each location, there are two images looking sideways from the vehicle trajectory.

out of every four images uniformly to create our smaller dataset. Figure 4.2 show sample images from this dataset. We refer to panoramic images and calibrated images from this dataset as GPP and GPS respectively.

We have yet another dataset of images gathered from Google Street View. The images are selected randomly from within streets of the city of Edmonton. This dataset contains 220K SIFT features. We refer to this dataset as *Auxiliary SIFT* dataset (SIFT Aux in short).

The last dataset, consists of 500K 128-dimensional vectors generated randomly on the positive coordinates of surface of a unit sphere. The vectors are generated simply by generating random points on the surface of a unit sphere and computing absolute value of individual coordinates of the points. We use this dataset to have some diversity in the distance distributions of our datasets.

4.2 Evaluation of Selectivity Prediction Method

Experiments of this section are designed to verify the correctness of the analysis about the performance of E2LSH presented in Chapter 3. More specifically, given E2LSH parameters L, K and W (or IE2LSH parameters m, K and W) and the overall distance distribution of the data points, we want to find the amount of error between the predicted selectivity and the actual selectivity of the E2LSH (or IE2LSH) tables. The experiments of this section confirm the correctness of our analysis.

4.2.1 Experiment Setup

In Chapter 3 we showed how one could relate the expected number of candidates to the distance distribution of the data points. Here, we set the E2LSH parameters and then compare the analytic results with empirical results to see how close our predictions are. An accurate prediction is required for setting the parameters optimally.

We use Rand and GPP datasets in these experiments. For each dataset, we compute pairwise distances of the points by randomly selecting two points from all the points at each step. There were 500K pairwise distances computed for the SIFT GPP dataset and 500K pairwise distances for the Rand dataset. Figure 3.8 and Figure 3.9 show the distance distributions for the Rand and SIFT datasets. We use these empirical distance distributions to estimate the number of candidates that we expect to retrieve from E2LSH tables.

We test different parameters to see how well our equations predict the number of candidates. We try to test the parameter settings that cover major parts of the parameter space; however, we only test the parameter settings which yield a selectivity more than 0.01 so that the empirical results are accurate enough. If selectivity is too low, for example only one feature has to be retrieved for each query vector, then the variance of the empirical results on our dataset will be too high.

4.2.2 Results and Conclusion

For each parameter setting, we compute the average and standard deviation of E2LSH tables' selectivity from 10 runs. The parameters, the actual and estimated selectivity of the E2LSH tables are shown in Table 4.1. Table 4.2 shows the same statistics for the case of IE2LSH. The last column of the tables shows the error ratios between the predicted and actual selectivities. These errors are computed from:

$$err(p1, p2) = \frac{p1 - p2}{max(p1, p2)} \tag{4.1}$$

In some cases, the error ratio goes as high as 20 percent. The error ratio has direct relation with the variance of the selectivity of the LSH structures (see highlighted rows in Table 4.1 and Table 4.2). This behavior is expected. Also, high variance in selectivity of the LSH structs in not good in practice. It is useful to consider these variances when selecting the parameters. Currently we only optimize the parameters to get the maximum success probability. The results show that our the predictions are above 90 percent accurate for most of the tested parameters both for E2LSH and IE2LSH. This amount of error is negligible considering that this is the error in the number of candidates not in the size of the dataset. The number of candidates is usually orders of magnitude smaller than the size of the dataset. The maximum effect these error can have on the running time is adding 10 percent to the search time of the LSH (t_s). By cutting the candidate list at the maximum number of candidates that we want to receive, we can get hard constraints on running time.

Table 4.1: Empirical and Estimated selectivity of E2LSH structures

Dataset	Parameters (W , K , L)	Estimated Selectivity		Actual Selectivity	Error
		Bounds	Exact		
SIFT	W:0.1 K: 2 L: 1	0.032	0.158	0.149 ± 0.019	0.060
SIFT	**W:0.5 K:10 L: 1**	**0.075**	**0.199**	$\mathbf{0.260 \pm 0.137}$	**0.232**
SIFT	W:0.5 K:16 L: 1	0.017	0.077	0.071 ± 0.048	0.077
SIFT	W:0.1 K: 2 L: 6	0.172	0.636	0.649 ± 0.022	0.020
SIFT	W:0.5 K:10 L: 6	0.365	0.726	0.751 ± 0.112	0.032
SIFT	W:0.5 K:16 L: 6	0.095	0.375	0.398 ± 0.083	0.058
SIFT	W:0.1 K: 2 L:21	0.468	0.965	0.968 ± 0.006	0.003
SIFT	W:0.2 K:16 L:21	0.0004	0.023	0.025 ± 0.004	0.085
SIFT	W:0.5 K:16 L:21	0.284	0.788	0.801 ± 0.072	0.016
RAND	W:0.1 K: 2 L: 1	0.056	0.214	0.213 ± 0.010	0.006
RAND	**W:0.5 K:10 L: 1**	**0.122**	**0.268**	$\mathbf{0.341 \pm 0.202}$	**0.213**
RAND	**W:0.5 K:16 L: 1**	**0.035**	**0.122**	$\mathbf{0.158 \pm 0.107}$	**0.226**
RAND	W:0.1 K: 2 L: 6	0.290	0.763	0.761 ± 0.007	0.002
RAND	W:0.5 K:10 L: 6	0.539	0.846	0.862 ± 0.073	0.019
RAND	W:0.5 K:16 L: 6	0.190	0.540	0.559 ± 0.150	0.034
RAND	W:0.1 K: 2 L:21	0.695	0.995	0.991 ± 0.000	0.004
RAND	W:0.2 K:16 L:21	0.001	0.065	0.075 ± 0.013	0.125
RAND	W:0.5 K:16 L:21	0.518	0.933	0.945 ± 0.029	0.013

Table 4.2: Empirical and Estimated selectivity of IE2LSH structures

Dataset	Parameters (W, K, m)	Estimated Selectivity		Actual Selectivity	Error
		Bounds	Exact		
SIFT	W:0.1 K: 2 M: 4 (L:6)	0.144	0.516	0.518 ± 0.034	0.004
SIFT	W:0.5 K:10 M: 4 (L:6)	0.296	0.597	0.610 ± 0.210	0.021
SIFT	**W:0.5 K:16 M: 4 (L:6)**	**0.082**	**0.305**	$\mathbf{0.354 \pm 0.212}$	**0.139**
SIFT	W:0.1 K: 2 M: 7 (L:21)	0.348	0.829	0.829 ± 0.014	0.000
SIFT	W:0.2 K:16 M: 7 (L:21)	0.0004	0.020	0.023 ± 0.009	0.122
SIFT	W:0.5 K:16 M: 7 (L:21)	0.216	0.606	0.596 ± 0.167	0.016
RAND	W:0.1 K: 2 L: 4	0.238	0.629	0.629 ± 0.014	0.001
RAND	W:0.5 K:10 L: 4	0.434	0.714	0.721 ± 0.237	0.010
RAND	W:0.5 K:16 L: 4	0.159	0.435	0.489 ± 0.174	0.111
RAND	W:0.1 K: 2 L: 7	0.517	0.909	0.882 ± 0.005	0.030
RAND	W:0.2 K:16 L: 7	0.001	0.055	0.063 ± 0.025	0.113
RAND	W:0.5 K:16 L: 7	0.382	0.763	0.831 ± 0.095	0.082

4.3 E2LSH Parameter Settings

In this section, we test our parameter selection method. We set the parameters for a dataset of SIFT features using the optimization method of Section 3.6. We also set the parameters by the approach

Datar *et al.* proposed in [2]. In addition, we do the nearest neighbor search search using KD-trees. We study the success rate of nearest neighbor search and the query times for E2LSH, IE2LSH and KD-Tree methods.

4.3.1 Experiment Setup

We use a dataset of 500K SIFT features as our dataset and another set of 1000 SIFT features as the queries. All the features were randomly selected from the GPS dataset. For each query feature, we find all the nearest neighbors that reside within distance 0.4 using all the methods. For ground truth, we find the true nearest neighbors using exhaustive search. We report the average query time and the recall of NN search for each method. Precision is not meaningful in our case because in case of LSH, the linear search part of the algorithm guarantees that we always get a precision of 1.

For kd-trees, there are 3 parameters: the number of trees (T), maximum number of comparisons (MC) and the number of NNs to retrieve (n). We set n to 50000 so that we are guaranteed to receive all the found NNs for each query feature. The maximum number of near neighbors for a query feature is roughly 5K in our dataset. For T and MC we try different values to get the best results. We tried MC values of 1000, 4000, 8000 and 16000. For T we used 1, 4 and 8 and 16. The program used 1.3 gigabytes of memory with 16 trees which is near our memory limit. KD-Trees support both range queries (in which we have a distance of interest and we are concerned only with near neighbors within that distance) and nearest neighbor queries. We use the range query from the FLANN [24] implementation.

E2LSH package has a script for selecting parameters. We run that script with the dataset SIFT features. As the success probability we used 90 percent so that we have roughly the same recall as the best performance of KD-Tree. We use those parameters with our data structures and report the running time and success rate. Our experiments with E2LSH package (E2LSH Org) showed that their code is more optimized that ours and given the same parameters, their code runs faster than our implementation by a factor of 2 to 3. This shows that the performance of our code (E2LSH, IE2LSH) can get better than what we report here by code optimization.

We also used two best parameter settings found by our method (named E2LSH in Table 4.3) after setting the time constraint to 10 milliseconds. We plot graphs similar to Figure 3.12 and choose the two curves that have higher peaks than the rest of the curves. We select parameters that correspond to the peaks of those two curves.

4.3.2 Experiment Results and Conclusion

Table 4.3 shows the results of the experiments. For the same recall, the running time of KD-Tree is slower than E2LSH by a factor of 2 to 3. We have used the two best parameter settings we could find with our method. The success rate of E2LSH nearly matches the predicted probabilities of Figure 3.13a.

Table 4.3: Results of NNS on SIFT features

Algorithm	Parameters	Search Recall (%)	Average Query Time (ms)
KD-Tree	T:1 - MC:16000	95	28
KD-Tree	T:4 - MC:16000	97	34
KD-Tree	T:8 - MC:8000	91	20
KD-Tree	T:8 - MC:16000	98	40
KD-Tree	T:16 - MC:8000	82	11
E2LSH	L:170 - K:12 - W:0.1	99	9.5
E2LSH	L:193 - K:16 - W:0.125	98	6.7
IE2LSH	M:22 (L=231) - K:18 - W:0.15	93	10.7
IE2LSH Org	M:22 (L=231) - K:18 - W:0.15[1]	93	3.5

Our experiments in this part show that our approach for setting E2LSH parameters is effective on real world datasets. The parameter selection in our case does not need empirical estimation of query time and therefore is more robust and less expensive computationally compared to previous methods. In the next section, we run LSH on two urban datasets for loop closure detection as an immediate use of our method.

4.4 Loop Closure Detection Experiments

We tackle the problem of visual loop closure detection by relying on LSH for fast nearest neighbor search and matching of individual visual features. Figure 4.3 shows our system at high level. First, we extract visual features (SIFT features) from the new image. We lookup each feature in the LSH tables and form a small (compared to database size) candidate list as the putative matching features for the query features. We then select some of the features as the true matching features based on their distance to the query feature and an adaption of the distance ratio criterion. We compute similarity of query image and all the keyframe images based on the matched features with respect to total number of features and select loop closing images by thresholding the similarity.

In this set of experiments, we run the system we described in previous chapter on both GPP and CC datasets to detect loop closures. We also try the BOW approach coupled with feature matching on the datasets as a baseline. As we mentioned before, BOW is the dominant approach for loop closure detection in visual SLAM applications.

In the following sections, we provide details about how we ran the experiments and the parameters that we used for both our approach and the BOW approach. The results of the experiments are presented in Section 4.4.3. All the experiments are run on the same machine.

[1]In the original E2LSH package, the random vectors are not normalized and we have scaled their W to match ours.

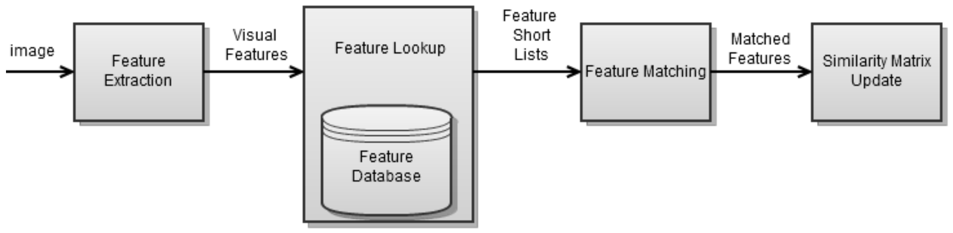


Figure 4.3: Overview of our loop closure detection system.

4.4.1 The Bag-of-Words Approach

We used the BOW approach with tf-idf heuristic for updating the similarity matrix of the images. The tf-idf weighting of scores comes from the text retrieval literature. *tf* is the frequency (number of repetitions) of a word (here visual word) in a document (image). A higher tf for a word w in a document means that word w is more important in the current document. Inverse document frequency (idf) is a measure of how important a word is in the whole dataset. If a word is present in every document, then that word is not distinctive and less useful for finding similar documents. idf is calculated by:

$$idf_w = idf(w) = \log \frac{N}{n_w} \tag{4.2}$$

where N is the number of documents in the dataset and n_w is the number of documents that contain word w. As the number of documents containing visual word w increases, the inverse document frequency of w approaches 1. On the other side, as n_w gets smaller, idf_w approaches $\log N >> 1$ In tf-idf scoring scheme, for each visual word that is present in the query image, the score for all the images that have that visual word is updated according to:

$$tf_{qw} idf_w = tf_{qw} \log \frac{N}{n_w} \tag{4.3}$$

and the similarity of two images p and q is given by:

$$sim(q,p) = \sum_{i=0}^{f_q} tf_{qw_i} idf_{w_1} = \sum_{i=0}^{f_q} tf_{qw_i} \log \frac{N}{n_{w_i}} \tag{4.4}$$

After the similarity between the query image and all the keyframes of the dataset are computed, the top few keyframes are selected as the candidate loop closing images. These candidates are then verified by direct feature matching. Our experiments showed that adding multiview geometry verification does not improve the loop closure detection performance significantly and we did not use MVG verification in our experiments.

The main parameter that BOW approach has is the number of visual words that are used to cluster the visual features in the images. It is common to use around 10K visual words [12]. We use 1000, 5000 and 10000 visual words in our experiments. We could not run clustering algorithms on our machines for more than 2500 cluster centers. Therefore, we used the pre-compiled SIFT visual vocabularies.

For the other parameters, we tried different values and we report only the ones that produce the best results. For feature matching in the candidate images, we used a threshold of 2 for the distance ratio criterion. The threshold for accepting images as loop closures was set to 0.08.

4.4.2 E2LSH

We are usually able to get a few sets of parameter values that satisfy the given time and space constraints for E2LSH. We set the time constraint so that each feature could be looked up in under 5 milliseconds (each image in less than 0.3 seconds on average). The parameters that we have used for E2LSH are listed in Table 4.5. Similar to what we did for the BOW approach, we try different threshold values for the distance ratio and similarity for accepting loop closures and report the best results.

By creating E2LSH tables, we are able to get a short list of candidate matches for each query feature. Next, we need to decide which ones actually match the query feature. Two features should match only when they come from the same object. In such cases, the features are closer to each other compared to features that come from different objects. The simplest option would be to threshold the euclidean distance between the features. We use the same threshold that we selected in the previous subsection.

Here we use the distance ratio technique with a simple modification. When we match features of a pair of images, we are guaranteed a unique match (if any) for each query feature. However in our case, where we may have multiple matching features, we need to allow for multiple matches. To achieve this, we sort the candidate features in the ascending order of distance from the query vector. Next we compute the distance ratios between each consequent pair of candidate features. If the ratio is below the threshold at any point, all the features before that point are accepted as matching features.

Figure 4.4 demonstrates how we match features using distance ratio. The red point is the query point and the other points are points from our dataset. Table 4.4 shows the distances to dataset points in the sorted order and the distance ratios for every consecutive pair of distances. Assume that the threshold on the distance ratio is 0.5. The third distance ratio is below the threshold, so the first three points (drawn with blue color) are accepted as matching points for out query.

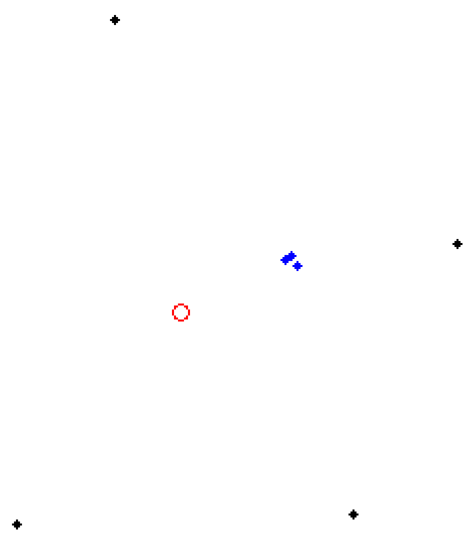


Figure 4.4: Demonstration of distance ratio for matching multiple features. Blue points show the points in dataset and the red point is the query point.

Table 4.4: Demonstration of distance ratio for matching multiple features. Distances and distance ratios for the points shown in Figure 4.4.

Distances	0.36	0.39	0.39	0.82	0.82	0.88	0.92
Distance Ratios	0.93	1.00	**0.48**	0.99	0.94	0.95	N/A

Table 4.5: Running Times for the City Center dataset

Algorithm	Parameters	Running Time[2]
BOW	C: 1000	0.04
BOW	C: 5000	0.19
BOW	C: 10000	0.38
LSH 1 DT	L:195 K:13 W:0.1	1.12
LSH 2 DT	L:200 K:15 W:0.125	1.21
LSH 1 DR	L:195 K:13 W:0.1	0.82
LSH 2 DR	L:200 K:15 W:0.125	0.69

4.4.3 Results and Conclusion

Table 4.5 and Table 4.6 show the running times for different methods. The running time of the BOW approach is better than our approach by a factor of 2 to 10 depending on the parameters. Our method spends under two second for each image which is sufficiently fast for real world applications.

Table 4.6: Running Times for the Google Pits side view dataset

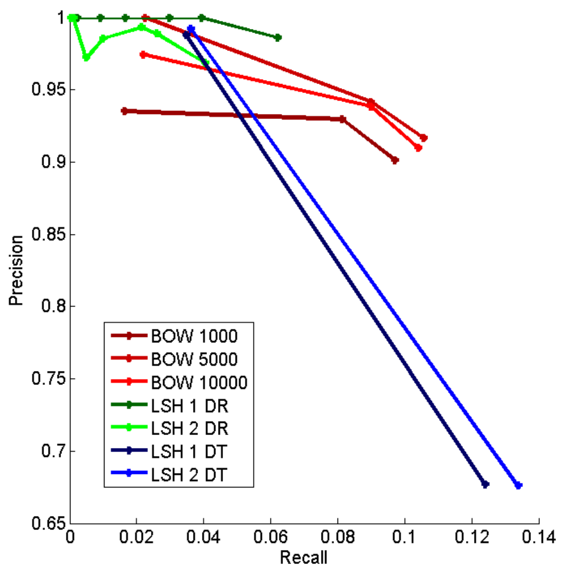
Algorithm	Parameters	Running Time[3]
BOW	C: 1000	0.21
BOW	C: 5000	0.45
BOW	C: 10000	0.72
LSH 1 DR	L:195 K:13 W:0.1	1.81
LSH 2 DR	L:200 K:15 W:0.125	1.74

Figure 4.5 shows precision-recall curves for different methods on our two datasets. In the City Center dataset, for precisions above 0.9, the recall of the BOW runs goes up to 0.12. The best performance is achieved with 5000 visual words which means that the performance of the BOW approach will not get better if we use a larger vocabulary. For E2LSH and distance ratio (LSH DR), our runs have very high precision but recall goes up to around 0.08. For E2LSH and absolute distance threshold (LSH DT), the performance is good for high precisions, but it drops dramatically as the threshold increases. In the experiments on the GPS dataset, the recall of the BOW approach goes up to 0.4 for precisions above 0.9. The performance of our LSH DR algorithms lies withing the same range.

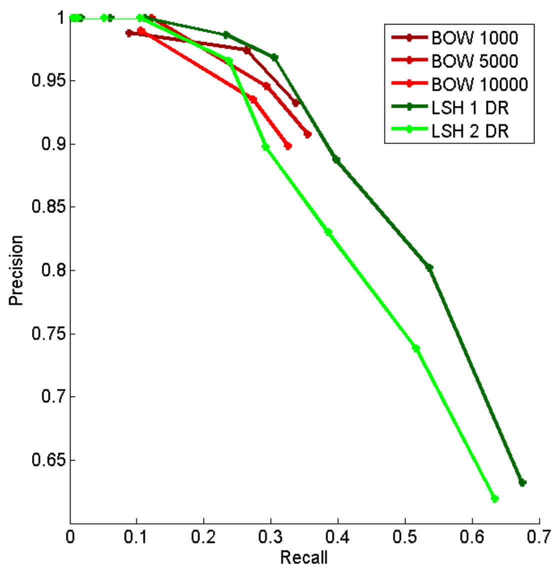
For the very high precision parts, which are the most important parts for SLAM applications, our LSH scheme gives good results. Figure 4.6 shows the map of the City Center dataset with the image locations. The red lines connect the loop closing images using ground truth and the green lines are the loop closing locations found by LSH DR 2. There are only two false positives in 318 detected loop closures. This high number of detected loop closures is sufficient for aligning the maps if there are drifts in the robot odometry information. Figure 4.7 shows the map of the Google Pits dataset with the loop closures that are detected by our LSH DR 2 algorithm. Again, there are a few loop closures detected at every sequence of loop closing images, which enables us to accurately align the locations.

[2]seconds per image

[3]seconds per image



(a) Precision-recall curves for the City Center dataset



(b) Precision-recall curves for the Google Pits side view dataset

Figure 4.5: Precision-recall curves for (a) the City Center dataset and (b) the Google Pits side view dataset. The points in each curve correspond to different thresholds for accepting loop closing images.

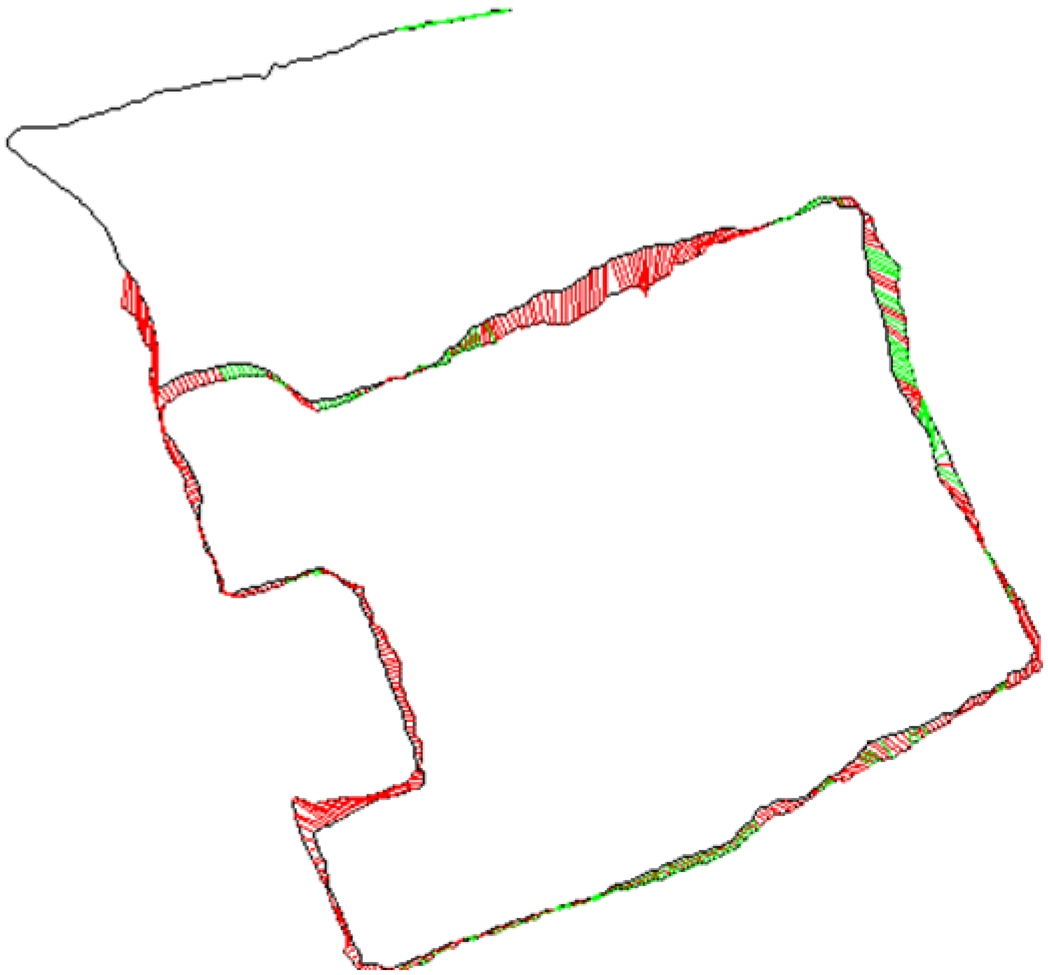


Figure 4.6: Map of the City Center dataset with the loop closures drawn. The red lines are the actual loop closures and the green lines are the loop closures found by the LSH DR 2 configuration of our algorithm. 198 loop closures are detected out of around 1K loop closures.

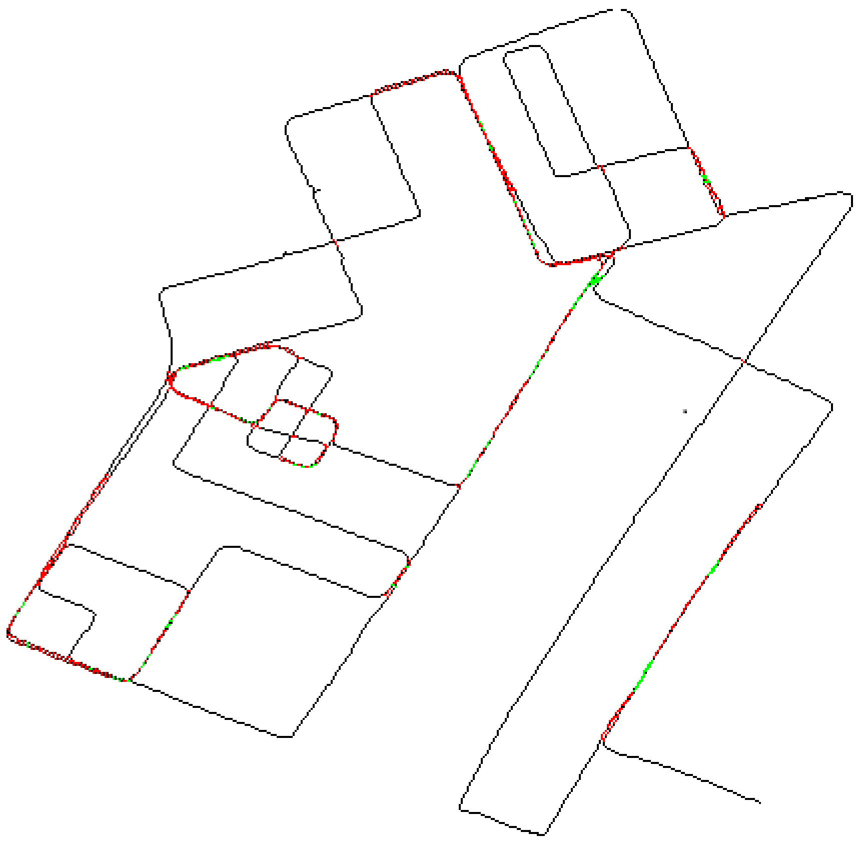


Figure 4.7: Map of the Google Pits dataset with the loop closures drawn. The red lines are the actual loop closures and the green lines are the loop closures found by the LSH DR 2 configuration of our algorithm.

Chapter 5

Conclusions and Future Work

In this work, we used distance distribution of data to predict the expected selectivity of E2LSH tables. Consequently, we used those predictions to find optimal parameters for E2LSH given the time and space constraints. It is possible to optimize for space requirement or running time if one needs to find parameters for a fixed success rate. We ran experiments on sample SIFT datasets and randomly generated data to show effectiveness of our method. Our experiments show that the distance distribution of points are stable for SIFT features and random points.

Additionally we ran some visual SLAM experiments, using E2LSH for finding similar features. We used a slightly modified version of the distance ratio technique to match features. The results of loop closing detection experiments show that E2LSH can perform as good as BOW approach coupled with direct feature matching which is the dominant approach for loop closure detection. Our approach has the benefit of requiring smaller statistics about the data points. In case of the BOW approach, one needs to create visual vocabularies from visual features while in our case, the overall distance distribution of the data points suffices.

In our work, we only considered E2LSH and IE2LSH schemes. It is known that both approaches require a lot of space compared to other nearest neighbor search algorithms. Some improvements have been made to E2LSH scheme to save space (see Section 2.3). It is possible to extend this work to predict the expected selectivity of LSH tables in other variations of E2LSH (Multi-probe LSH for example) and set parameters automatically for those schemes.

For the SLAM experiments, we only used SIFT features because of their superior distinctiveness compared to other visual features. It is possible to use other types of features with LSH. We showed empirically that the distance distributions were stable for SIFT features over different datasets. This has to be shown for other types of visual features.

Another aspect of our work that can be improved is the parameter selection method. We only compute the expected number of candidates and minimize the expected query time. However, the variance of this time is important especially for real-time applications.

Bibliography

[1] Alexandr Andoni. *Nearest Neighbor Search: the Old, the New, and the Impossible*. PhD thesis, 2009.

[2] Alexandr Andoni and Piotr Indyk. E2lsh 0.1 user manual, 2005.

[3] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In Ale Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision ECCV 2006*, volume 3951 of *Lecture Notes in Computer Science*, pages 404–417. Springer Berlin / Heidelberg, 2006.

[4] Jon Louis Bentley. K-d trees for semidynamic point sets. In *Proceedings of the sixth annual symposium on Computational geometry*, SCG '90, pages 187–197, New York, NY, USA, 1990. ACM.

[5] O. Booij, B. Terwijn, Z. Zivkovic, and B. Krose. Navigation using an appearance based topological map. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3927 –3932, april 2007.

[6] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *European Conference on Computer Vision*, pages 778–792, 2010.

[7] J. A. Castellanos, R. Martnez-cantn, J. D. Tards, and J. Neira. Robocentric map joining: Improving the consistency of ekf-slam. *Robotics and Autonomous Systems*, 55:21–29, 2007.

[8] O. Chum, J. Philbin, and A. Zisserman. Near Duplicate Image Detection: min-Hash and tf-idf Weighting.

[9] Ondrej Chum, Michal Perdoch, and Jiri Matas. Geometric min-hashing: Finding a (thick) needle in a haystack, 2009.

[10] Paolo Ciaccia and Marco Patella. Using the distance distribution for approximate similarity queries in high-dimensional metric spaces. *Database and Expert Systems Applications, International Workshop on*, 0:200, 1999.

[11] Paolo Ciaccia, Marco Patella, and Pavel Zezula. A cost model for similarity queries in metric spaces. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '98, pages 59–68, New York, NY, USA, 1998. ACM.

[12] Mark Cummins and Paul Newman. Fab-map: Probabilistic localization and mapping in the space of appearance. *The International Journal of Robotics Research*, 27(6):647–665, 2008.

[13] Mark Cummins and Paul Newman. Appearance-only slam at large scale with fab-map 2.0. *The International Journal of Robotics Research*, 2010.

[14] M. Datar and P. Indyk. Locality-sensitive hashing scheme based on p-stable distributions. In *In SCG 04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM Press, 2004.

[15] Uriel Feige and Gideon Schechtman. On the optimality of the random hyperplane rounding technique for max cut. *Random Struct. Algorithms*, 20:403–440, May 2002.

[16] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[17] Antonin Guttman. Readings in database systems. chapter R-trees: a dynamic index structure for spatial searching, pages 599–609. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[18] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.

[19] Herve Jegou, Laurent Amsaleg, Cordelia Schmid, and Patrick Gros. Query adaptative locality sensitive hashing. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 825–828, 2008.

[20] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60:91–110, November 2004.

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

[22] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe lsh: efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 950–961. VLDB Endowment, 2007.

[23] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27:1615–1630, 2005.

[24] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In Alpesh Ranchordas and Helder Arajo, editors, *VISAPP (1)*, pages 331–340. INSTICC Press, 2009.

[25] P. Newman, D. Cole, and K. Ho. Outdoor slam using visual appearance and laser ranging. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1180 –1187, may 2006.

[26] Paul Newman, Gabe Sibley, Mike Smith, Mark Cummins, Alastair Harrison, Chris Mei, Ingmar Posner, Robbie Shade, Derik Schroeter, Liz Murphy, Winston Churchill, Dave Cole, and Ian Reid. Navigating, recognizing and describing urban spaces with vision and lasers. *Int. J. Rob. Res.*, 28:1406–1433, November 2009.

[27] Rina Panigrahy. Entropy based nearest neighbor search in high dimensions. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, SODA '06, pages 1186–1195, New York, NY, USA, 2006. ACM.

[28] G. Sibley, C. Mei, I. Reid, and P. Newman. Adaptive relative bundle adjustment. In *Proceedings of Robotics: Science and Systems*, Seattle, USA, June 2009.

[29] R. Smith, M. Self, and P. Cheeseman. *Estimating uncertain spatial relationships in robotics*, pages 167–193. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[30] Christoph Strecha, Alex Bronstein, Michael Bronstein, and Pascal Fua. Ldahash: Improved matching with smaller descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 99(PrePrints), 2011.

[31] Kengo Terasawa and Yuzuru Tanaka. Spherical lsh for approximate nearest neighbor search on unit hypersphere. In Frank K. H. A. Dehne, Jrg-Rdiger Sack, and Norbert Zeh, editors, *WADS*, volume 4619 of *Lecture Notes in Computer Science*, pages 27–38. Springer, 2007.

[32] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 194–205, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

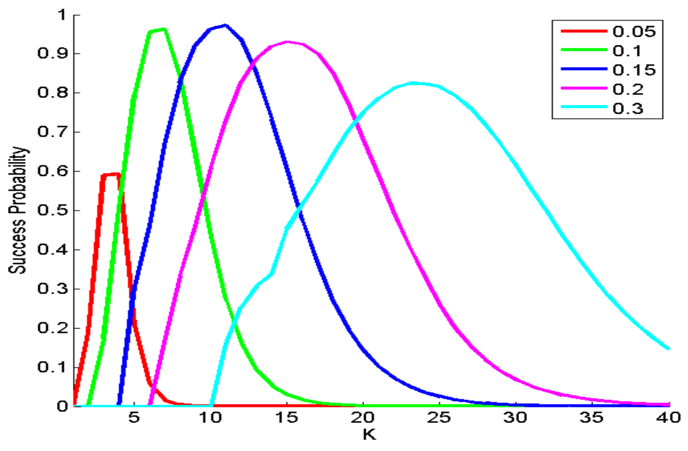
[33] Hong Zhang, Bo Li, and Dan Yang. Keyframe detection for appearance-based visual slam. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2071 –2076, oct. 2010.

Appendix A

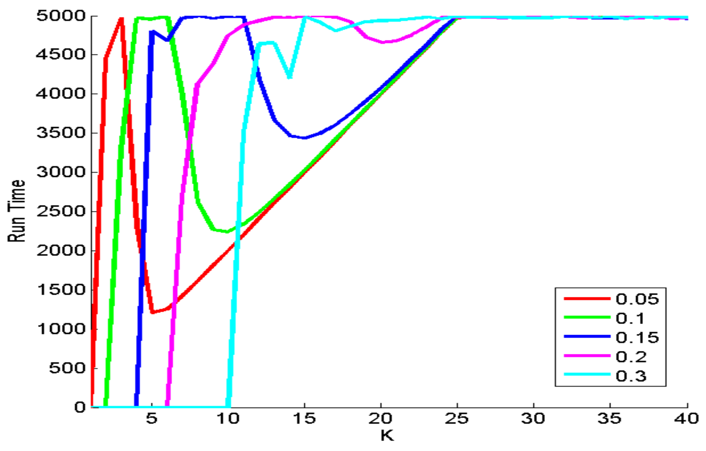
Appendix 1: Plots of E2LSH using Bounds on the Sphere Caps

We showed in Section 4.2 that the selectivity estimated by using bounds on the sphere caps is very inaccurate. The selectivity of hash functions is sometimes two orders of magnitude smaller than the actual selectivity. Here we present the performance graphs for E2LSH by using the bounds just for the sake of completeness. In practice the performance of E2LSH is very close to exact estimations that we presented in Section 3.6 and used in our second and third set of experiments.

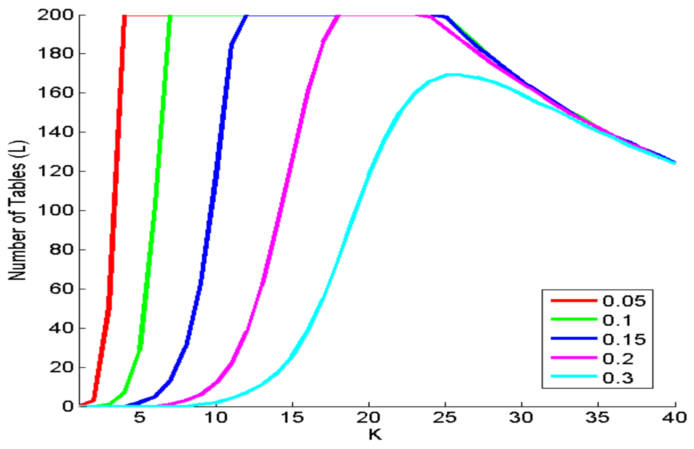
Figure A.1 and Figure A.2 show performance graphs for E2LSH and IE2LSH respectively. Because the selectivity of hash functions is under estimated when we use bounds, the number of tables goes up more sharply compared to when the cap areas are computed exactly. Peak of success probability is higher compared to the actual values and it occurs at a lower value of K. This is caused by the fact that the underestimation of the selectivity is more than the under estimation of the success probability.



(a) Success probability as a function of W and K

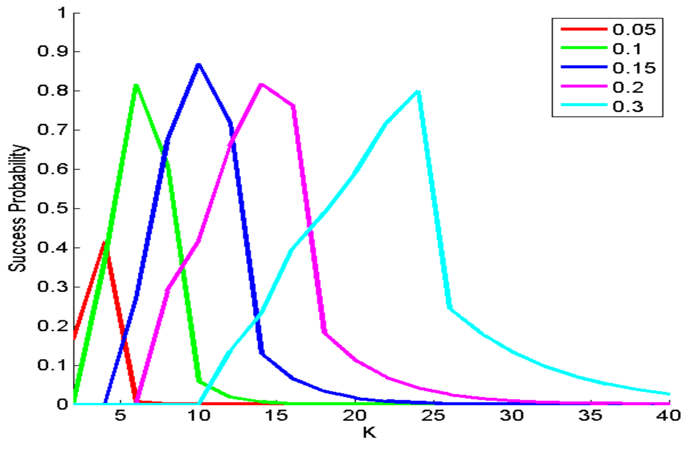


(b) Running time as a function of W and K

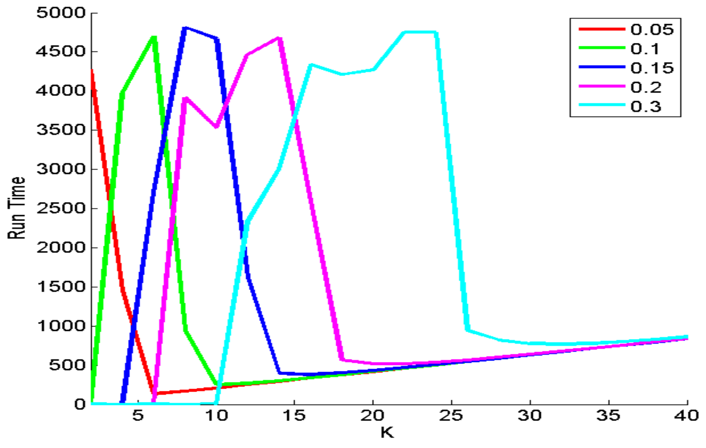


(c) Optimal L value as a function of W and K

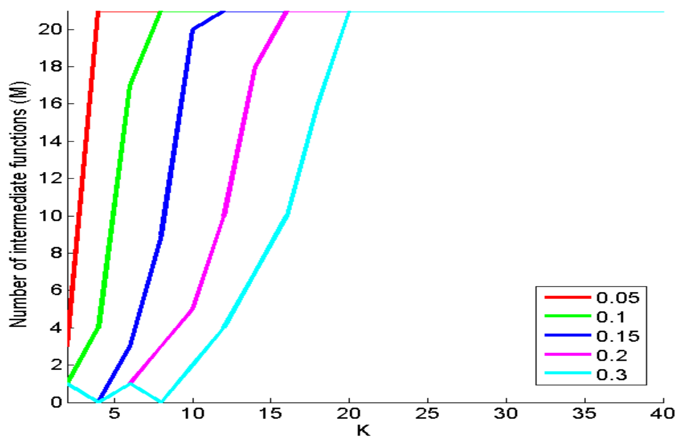
Figure A.1: The (a) success probability, (b) running time and (c) L optimal value as functions of W and K for E2LSH. The time limit was set to 400K operations (3K vector dot products). The maximum number of tables was set to 200 (roughly 3GB).



(a) Success probability as a function of W and K



(b) Running time as a function of W and K



(c) Optimal L value as a function of W and K

Figure A.2: The (a) success probability, (b) running time and (c) L optimal value as functions of W and K for IE2LSH. The time limit was set to 400K operations (3K vector dot products). The maximum number of tables was set to 200 (roughly 3GB) which implies that $m \leq 21$.