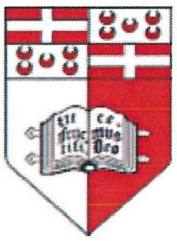
White Cane Device: A Mobile Assistant for Visually Challenged People

Judie Attard

Supervisor: Dr. Matthew Montebello



Faculty of ICT

University of Malta

27th May 2011

Submitted in partial fulfilment of the requirements for the degree of B.Sc I.C.T. (Hons.)

To all those persons who in some way or another made me the person I am today, whether we crossed paths for a while or are simply walking together in the same direction.

ACKNOWLEDGEMENTS

First and foremost I would like to express my deepest gratitude to my tutor Dr. Matthew Montebello. I would like to thank him for helping me whenever I required any guidance, even out of office hours. His support was vital for the completion of this thesis.

I would like to extend my thanks to Mr. Stanley Debono and Mr. Michael Micallef from the Foundation for IT Accessibility (FITA) for believing in my work and providing me with important feedback for the evaluation of White Cane Device.

My thanks also go to my parents, Frances and Joseph, and my brother, Michael-George, who always believed in me. Even through rough moments, they always encouraged me and supported me, both in life and in education.

Special thanks go to my friends. I would like to thank my closest friends for being such, and for all the times we spent together; laughing, crying, or even both at once. I would also like to thank a relatively new friend, Fernand Fenech, for the designing of the icon used for the White Cane Device.

Last but not least my thanks go to a special friend, Jeremy Debattista, and his parents, Michelle and Ronnie, who welcomed me with open arms in their home and treated me as their own. Jeremy gave me help and suggestions invaluable for the development of this thesis. He encouraged me when everything seemed bleak, and kept me company through endless hours of coding and documentation. "V".

ABSTRACT

A large percentage of the population is affected in various ways by visual impairments and most of the latter have no effective cure. Such visual impairments can significantly deteriorate the quality of life of people affected. The goal of this study is to exploit the portability and availability of mobile phones to provide a means of guidance to the visually impaired while travelling.

The proposed system takes into consideration a visually impaired person while travelling through the city. The main challenge for such a person is to continously identify the correct location of his/her whereabouts. This system assists the user, through image recognition from a mobile phone, by guiding and offering information of the immediate vicinity. The 'White Cane Device' being proposed enables the user to capture a picture, and receives back information about the surroundings. This includes the object recognized within the query image, as well as the current location of the user if s/he is some distance away from the object recognized within the image.

The White Cane Device was implemented in a client-server architecture, where the client acts as a peripheral device, and the server implements the SIFT feature extraction algorithm to perform object recognition. The GPS coordinates of the images are used to optimize the latter process. This is done by reducing the search space. A thorough evaluation of the proposed system gave exciting results and showed that White Cane Device performs reliable matching even when images contain occlusion or are taken from various perspectives. The tested system returned results within more than acceptable time spans and received positive feedback from the usability evaluations held.

CONTENTS

LIST OF FIGURES

LIST OF EQUATIONS

Chapter 1

INTRODUCTION

A large percentage of the population is affected in various ways by visual impairments and most of the latter have no effective cure. Such visual impairments can significantly deteriorate the quality of life of people affected. With advancement in technology, various products offering aid in daily activities are available on the market, however these are often expensive and/or dedicated to a single task. For example, NanoPac [1] offers several software solutions such as screen enlargers, OCR, screen readers and the KNFB mobile reader.

1.1 MOTIVATION

The goal of this thesis is to exploit the portability and availability of mobile phones to provide a means of guidance to the visually impaired while travelling. The imaging, processing and communication capabilities of mobile phones provide the device with exciting possibilities for new uses such as image processing and computer vision technologies. Being a portable hand-held device, a camera enabled mobile phone has some unique advantages. Firstly, such devices are already carried by a very large number of people, and secondly, new functions can be added to the device easily without the requirement for any additional hardware. Besides, the communication capability of mobile phones, such as an Internet connection and GPS capability, provide the perfect setting for the purpose of functions such as the one explored in this thesis.

In the proposed system, we consider the problem of identifying the content of an image. This can be done using two methods; namely Content Based Image Retrieval (CBIR) or Object Recognition. Consider a visually impaired person travelling through a city. He feels lost and requires some guidance, so he takes his mobile phone, takes a picture of his surroundings and runs a system implementing one of the mentioned methods. CBIR focuses on the semantic annotation of images, rather than textual as per regular search engines. A CBIR system has a database module containing indexed samples of images of the relevant subject (in this case, images of the city in question). The picture taken by the visually impaired person is processed, and a similarity measure is used to find a similar image annotated about the image content from the database. Object recognition, on the other hand, involves identifying an instance of a particular object in an image. The database module, instead of containing images, would contain features extracted from the latter, and any other relevant data. In this work we build on the latter method.

1.2 AIMS AND OBJECTIVES

The main aim of this thesis is to provide a travelling aid to visually impaired persons. The system should enable the latter to take an image using a mobile phone equipped with a camera and return any objects (mainly buildings) recognized in the snapshot, such as a particular pharmacy. Also, the GPS capability of a phone should be exploited to enhance the object recognition process

by minimizing the image search space. To implement such a system, the following objectives have to be met:

- The development of a user-friendly mobile application enabling the user to take a geo-tagged image;
- The implementation of an object recognition method;
- The use of GPS coordinates to reduce the search space;
- The development of a system incorporating the mobile application and the object recognition method.

To obtain the mentioned objectives various challenges need to be overcome. The first challenge is the limited processing power of the mobile phone. Object recognition (and the required image pre-processing) are quite resource intensive. Thus an adequate solution is required as the user obviously expects relevant results in real time. Another challenge is the visual impairment of the users; one cannot expect the user to focus perfectly the mobile phone camera on the particular object of interest, therefore allowance for blurry images is a must. Also, one must keep in mind the relatively low-quality mobile phone camera lenses when compared with regular cameras. Another challenge to be considered is the change in lighting and pose. Images of a particular object taken in different lighting conditions and/or from a different viewpoint can have a large impact on the object recognition process. Clutter and occlusion also have a large impact on object recognition. The GPS capability of the mobile phone can be used in such a way that when the user takes an image, the relevant coordinates of the current position of the user are recorded. This data can also be used to find the current location of the user.

1.3 CHAPTERS' OVERVIEW

After having introduced the context of this thesis, in Chapter 2 we discuss relevant literature which sustains our work and gives the reader a background. In Chapter 3 we discuss the specification and design of the proposed system 'White Cane Device'. Chapter 4 explores the implementation details for the development of the system. Various experiments and evaluations were carried out and discussed in Chapter 5. Proposed future work is described in Chapter 6 while Chapter 7 concludes this thesis.

Chapter 2

BACKGROUND AND LITERATURE REVIEW

In this chapter we describe and explain some basic concepts and methods used in this thesis to help the reader better understand the content of the latter. These include the GPS system, image transformations, various object matching approaches and algorithms, and similar systems which implement the latter object matching algorithms.

2.1 GPS

The Global Positioning System (GPS) provides location information to a GPS enabled device if said device is within line of sight to GPS satellites orbiting the Earth. The conventional units used for GPS coordinates are degrees, minutes and seconds. The latter two are not to be confused with time units. GPS coordinates usually also include a letter denoting the orientation, namely N, S, E, or W. N and S denote the Northern and Southern hemisphere while the E and W denote the respective direction from the prime meridian.

GPS coordinates can be displayed in decimal or degree format:

- **Degree (DDD, MM, SS)**
  10° 44′ 29.97″ N, 119° 20′ 19.44″ W
- **Decimal (DD.DDDD)**
  10.741658, -119.338733

It is important to note that the decimal format does not contain seconds and four decimal places are usually enough. Also, W and S are denoted by a negative sign. The methods used to do the conversion from degrees, minutes and seconds (DMS) to decimal degrees, and vice versa, are the following [1]:

- **DMS to Decimal Degrees:**

$$d = D + \frac{M}{60} + \frac{S}{3600}$$

Equation 1: Equation to convert degrees from DMS to decimal format

- **Decimal Degrees to DMS:**
  Being somewhat complicated, let us explain this method using an example. Consider having the decimal degree reading of 5.23456 degrees. To convert it to DMS format, we first subtract the whole degree: 5.23456 – 5 = 0.23456. Then we multiply the remaining

[1] http://www.fao.org/docrep/006/y4816e/y4816e0e.htm

fraction with 60 minutes: 0.23456 * 60 = 14.0736. The whole number gives the minutes value: 14.0736 – 14 = 0.0736. Finally, the fraction is multiplied with 60 seconds: 0.0736 * 60 = 4.416, giving the remaining seconds. Thus the DMS format of 5.23456 degrees is 5 degrees, 14 minutes and 4.416 seconds.

The distance between two GPS coordinates can be calculated using the Haversine formula [2]. This equation gives the great-circle distance between two points on a sphere using the longitude and latitude coordinates. The Haversine formula is as follows:

$$dlon = lon2 - lon1$$

$$dlat = lat2 - lat1$$

$$a = \sin^2\left(\frac{dlat}{2}\right) + \cos(lat1) * \cos(lat2) * \sin^2\left(\frac{dlon}{2}\right)$$

$$c = 2 * \arcsin(\min(1, sqrt(a)))$$

$$d = R * c$$

Equation 2 : The Haversine Formula. Equation from [2].

Where *(lon1, lat1)* and *(lon2, lat2)* are the coordinates (in radians) of the two points and *R* is the radius of the Earth (6371km)[2]. The resulting distance *d* is in the metrics used for the Earth's radius.

2.2 Digital Image Representation

A digital image is represented by a two-dimensional grid of points known as pixels. Each pixel has a color value representing the color tone of that particular point on an image. This is the color depth of the image and can be represented using a number of bits. An image can be of various color depths, the most widely used being:

- 1-bit, which represent a monochrome (black and white) image
- 8-bits, which can represent either greyscale or 256-color images
- 24-bit, which represent the full colors possible by most computers

The image resolution is the number of pixels in the rows and columns of the grid representation of the image; the larger the resolution, the larger the file size. For example a 384x640 image with 24-bit color depth is approximately 100 Kb.

[2] http://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html

Figure 1: The same image in RGB and greyscale format. Image adapted from [48].

2.3 Image Transformations

2.3.1 Scale

Image scaling is the process of resizing a digital representation of an image. Scaling involves a finding a trade-off between efficiency, smoothness and sharpness. As the size of an image is increased, so do the pixels which make up the digital representation of the image. This results in the image having a stair stepped effect (pixelation). Conversely, reducing the size of an image results in the image appearing smoother and sharper.

Figure 2: Scale Change (Zooming). Images adapted from [33].

2.3.2 Rotation

Image rotation is the process of revolving an image around a point of rotation.

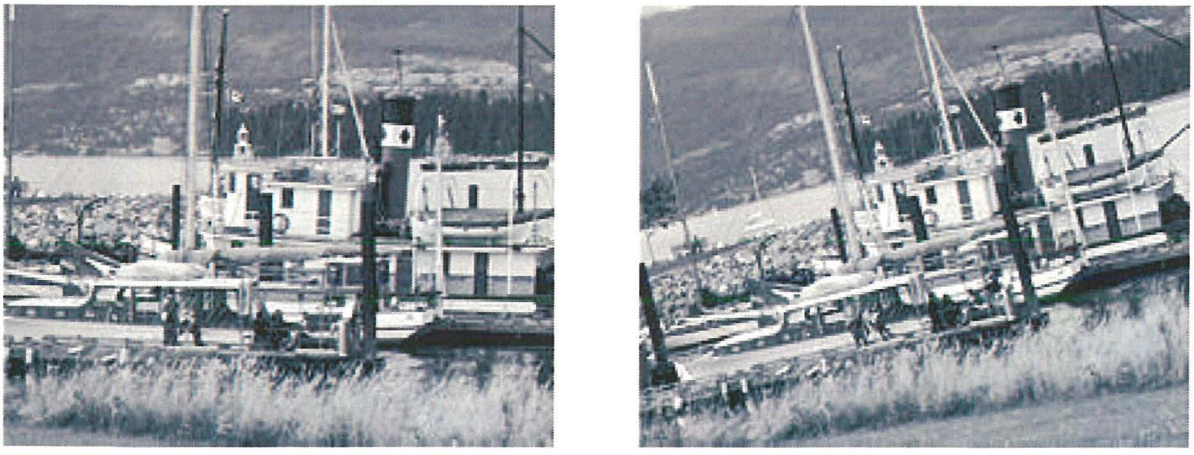


Figure 3: Image Rotation. Images from [33]

2.3.3 *Blur*

A smudged or unclear image is called a blurred image. The blurring of an image can be caused by the movement of either the subject of the picture or the movement of the camera if the subject is a static background.

Figure 4: Blurring. Images from [33].

2.3.4 *Illumination*

Illumination is the degree of light captured in an image. This depends on the lighting conditions in which the picture was taken or on camera settings.

Figure 5: Illumination change. Images from [33].

2.3.5 *Affine transformations*

A transformation changes the position of points on a plane. Such a transformation results in the points acquiring a different shape. An affine transformation can be defined as a transformation that fixes the origin in the image, while translating the points on the entire plane. An affine transformation is generally brought about by a viewpoint change.

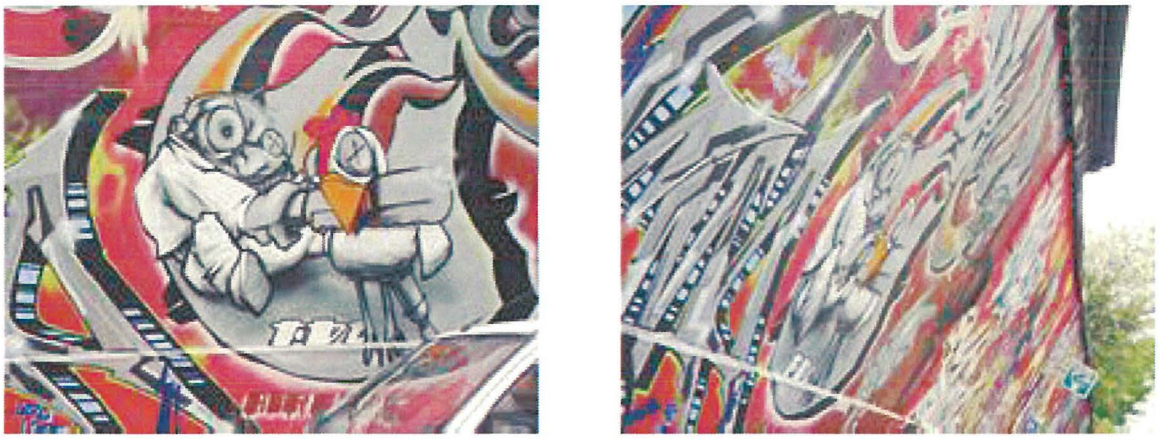


Figure 6: Affine Transformation. Images from [33].

2.4 Object Recognition

Object recognition, as stated by Yang in [3], is one of the most astonishing capabilities humans possess from a very young age. Simply by looking at an object, humans are able to categorize or identify it, despite any appearance variation due to change in illumination, pose, and any occlusion. Furthermore, after seeing some objects of the same category, humans are capable of categorizing objects that they have never seen before. Even if it is a daunting task to create systems which mimic human object recognition capabilities, several systems with scene and pattern recognition abilities have been developed. These are explored in Section 2.7.

The object recognition problem starts with one or more pictures of an object, possibly taken from various perspectives and using different cameras and/or resolutions. These pictures are the search images. Given a query image, the problem is whether the latter contains one or more objects contained in the search images. Object recognition differs from object categorization in that the problem is not to recognize a class of objects, such as cars or buildings, but to recognize an instance of a particular object. In terms of recognition, several challenges have to be tackled, the most conspicuous of which is the apparent deformation of an image due to a change in perspective. Other challenges include recognizing objects from different viewpoints, varying illumination, and occlusion by other objects and clutter. The problem of object recognition can be tackled using various approaches [3]. The latter are explored in this chapter.

2.4.1 Geometry-Based Approaches

Since the earliest attempts at object recognition, efforts were focused on the use of the geometric models of objects to account for their change in appearance due to perspective and illumination change. A number of reasons exist as to why geometry has performed such an important role in object recognition, such as viewpoint invariance, illumination invariance, as well as a well-developed geometric theory [4]. Alas, it has been shown that the reliable extraction of geometric primitives, such as lines and circles which are invariant to viewpoint change, is only possible under limited [3].

An early example of a geometry-based approach is the "Blocks world" framework (late 1950s). The aim of this approach was a simplification of object shapes in such a way that mathematical models could be applied to solve the recognition problem. Objects are simplified to polyhedral projection of such shapes into images under perspective. A powerful implementation of the

blocks world was that of Lawrence G. Roberts [5]. For more than a decade, the Blocks world framework was a main field in vision research before more realistic scenes took over. This was because it became obvious that such approaches would not hold in real-world scenes since too many assumptions were being made.

The generalized cylinder (GC), proposed by Thomas Binford [6], was the next major advancement in the field of representations for recognition. The main reasoning behind the generalized cylinder is that the majority of shapes can be expressed as a "sweep of a variable cross section along a curved axis" [4]. Gerald Agin and Rodney Brooks were Binford students who based their work on the generalized cylinder. Agin developed "A ranging system... [which] obtains three-dimensional images of curved solid objects. The object is segmented into parts by grouping parallel traces obtained from the ranging system. Making use of the property of generalized translational invariance, the parts are described in terms of generalized cylinders, consisting of a space curve, or axis, and a circular cross section function on this axis" [7]. Brooks, on the other hand, introduced "new approaches to prediction and interpretation based on the propagation of symbolic constraints" [8]. Practically, the system could prove theorems "regarding the existence of a parameterized GC configuration with associated tolerances" [4].

Object oriented representation is an approach based on the premise that objects can be recognized from their shape outlines and interior intensity discontinuity boundaries. In the 1970s a different representational scheme was introduced, based on a network of the diverse 2D views of a particular object. This scheme is called an aspect graph, the computation of which Jean Ponce and David Kriegman later extended to generalized cylinders [9]. Aspect graphs are made up of nodes for each aspect, and edges connecting adjacent aspects. In the early 1990s, aspect graphs met with major difficulties such as the rapid growth of size in graphs, as well as their complexity.

Perceptual grouping was the process which formed the foundation of the early geometric period. Perceptual grouping consisted of the notion that "bottom-up boundary descriptions could be formed from single intensity views of an object" [4]. The latter notion, however, was not problem-free. Some problems included:

- Occlusion by complex-textured objects;
- Clutter with high edge density in the background.

Such difficulties prompted researchers to implement recognition systems based on fragmentary feature segmentations in terms of 2D point and line or curve segments. Instead of the generic descriptions which predominated the early period, the organization of the features was based on a specific individual object model.

In the early 1980s, the focus was mainly based on 2D planar shapes or 3D objects as portrayed by 3D range cameras. It was felt that it was more important to solve 2D planar object recognition before attempting to solve the harder problem of 3D object recognition from a single intensity image. At this point in time, one can say that the 2D problem is solved for many daily-life problems such as industrial inspection; however a large amount of background clutter along with a significant amount of occlusion can result in a large number of false hypotheses.

Later on in the decade, recognition of 3D objects from 2D intensity images re-conquered the scene. Such approaches made use of viewpoint consistency where the pose of an object was calculated from a small number of features. Instead of bowing to the limitation of full-perspective image formation, affine image projection models were used.

At the end of the 1980s, the object recognition community showed a rise in interest in trying to automate the creation of models for recognition which would ideally only require a single view of an object. The concept of geometric invariance was explored, where viewpoint-invariant properties of an object are determined.

Due to the difficulties such as missing features and noisy geometry in feature segmentation methods, such methods did not advance much. Since the smallest possible number of image features are used for the invariant construction, geometric invariants were noise prone. Yet, in spite of these limitations, by 1995 it was possible to recognize a number of 3D objects in a cluttered scene [10] by "exploiting class-based invariance such as of surfaces of revolution and canal surfaces" [4]. However, this progress all led to the realization that more progress would depend on better image segmentation methods.

2.4.2 Appearance-based Algorithms

The development of state-of-the-art feature descriptors and pattern recognition algorithms has enabled efforts to be transferred from geometry-based object recognition to appearance-based techniques. The goal of object recognition is to recognize an object from an image containing other clutter. To aid this purpose, discriminative classifiers such as k-nearest neighbor, neural networks with radial basis function (RBF), dynamic link architecture, support vector machines (SVM), and boosting algorithms have been used to exploit the class-specific information, and to recognize 3D objects from 2D images [11] [12]. As opposed to geometry-based object recognition, appearance-based methods are more effective at handling viewpoint and illumination change, but somewhat less effective at handling occlusion.

Feature-based algorithms fall under the appearance-based algorithms category. In the fields of computer vision and image processing the term feature detection refers to methods which aim at extracting image information and making checking at each image point if there exists an image feature or not. This stage is known as feature detection. This approach is based on the idea of finding interest points which are invariant to change due to scale, illumination and affine transformation. As such, there does not exist a proper definition of a feature. Rather, the definition of a feature depends on the type of application at hand. Simply, a feature can be defined as an interesting part of an image, or a specific structure in the image itself such as points or edges. These interest points are then converted to feature descriptors, which describe the feature in some structured way. The descriptions may include information about the scale, translation, color, texture and rotation of the extracted feature. Such descriptors have been successfully implemented in several applications such as the recognition of object categories [13], [14], image retrieval [15], [16], wide baseline matching [17], object recognition [18], [19], texture recognition [20], video data mining [21], robot localization [22] and building panoramas [23].

Since such features are mostly used as starting points for subsequent algorithms, the resulting complete algorithm can only be as good as the implemented feature detector. Thus, desirable properties of features are:

- **Distinctiveness** – a distinctive feature can be matched reliably against a large database of features extracted from many images;
- **Invariance** – an invariant feature can provide "robust matching across a substantial range of affine distortion, change in 3D viewpoint, addition of noise, and change in illumination" [18];
- **Repeatability** – a repeatable feature is detected in two or more views of the same object or scene.

Many computer vision applications require feature detectors as an initial step, thus various feature detectors have been developed, varying on the kind of features detected, the computational complexity and the feature properties.

Once the detection of features is complete, data from the image is extracted for each feature, amongst which the feature descriptor and other details such as the coordinates of the feature on the image. The descriptor contains data such as the scale and the orientation from which the feature was extracted. The number of dimensions a descriptor has impacts directly the time taken to compute the matching process which is often based on finding the distance between vectors, such as Euclidean distance or Mahalonobis.

Following are some popular state-of-the-art feature-based algorithms. It is important to note that they all operate on greyscale images.

2.4.2.1 SIFT

The SIFT method extracts distinctive invariant features from images which are then used to match different views of some object reliably. The latter features are scale and rotation invariant, and provide robust matching even with substantial noise addition, illumination differences and affine distortion. The features are also highly distinctive. The SIFT approach generates a large number of features from all over the image and over a large range of scales and locations. This number of features is important for object recognition since the capability of detecting objects in cluttered scenes requires at least the correct matching of 3 features for each object, for reliable identification.

An important aspect of the SIFT method is that a large number of features is generated from an image, although the number depends on image content and various parameter choices. The number of features generated is especially important for object recognition, where the correct matching of at least 3 features for each object is required to detect small objects in cluttered backgrounds.

The main stages required to generate the set of image features are as follows [18]:

- **Scale-space extrema detection:**
  The first stage of this process searches through all scales and image locations. A difference-of-Gaussian function is implemented to find possible interest points which are scale and orientation invariant.

A scale space attempts to replicate the concept of the level of detail one notices when viewing an object. For example when viewing a tree, one might view the leaves in detail, or, on the other hand, one might view the tree as a whole, forgoing the details of the tree's leaves. The mathematical method of replicating this concept is by using the Gaussian blur. Simply put, Gaussian blur takes the original image and generates progressively blurred out images. Then the image is resized to half its size and the blurred out images are generated again. This process is repeated for as long as required. As can be seen in ***Figure 7***, images of the same size form an octave, where each octave is made up of increasingly blurred images.

Figure 7: Scale Space. Image from [49].

The created scale space is then used to create another set of images, namely the Difference of Gaussians (DoG). The DoG is used as an approximation to the scale-normalized, but computationally intensive Laplacian of Gaussian (LoG). For the DoG, two consecutive images in an octave are taken and one is subtracted from the other. This process is repeated for all the images in the octave, for all octaves, as can be seen in ***Figure 8***.

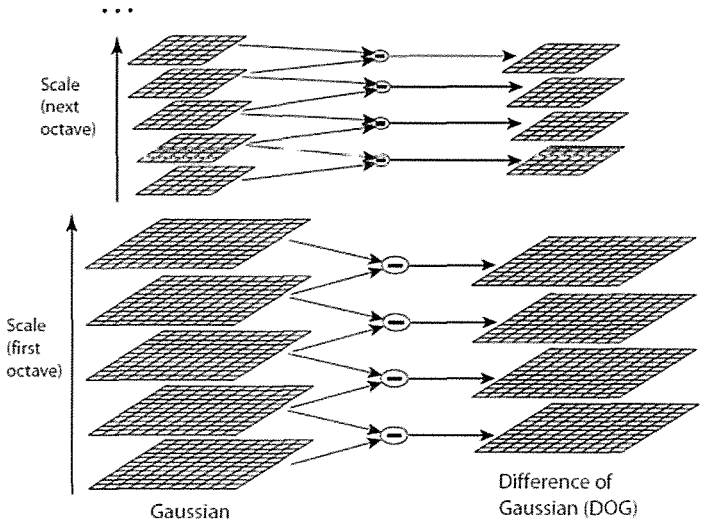


Figure 8: Difference of Gaussian. Image from [18].

- **Keypoint localization:**
  At each possible location, a detailed model is fit to determine location, scale and ratio of principal curvatures. Thus, points having low contrast (therefore noise-sensitive) or points localized along an edge are discarded. Keypoints are selected based on measures of their stability.

  The first step for this stage is the coarse location of maxima and minima in the DoG images. This is done by iterating through each pixel in the image and checking all its neighbors. This process is repeated within the current image, and also the scale above and below it. This is portrayed in ***Figure 9***.

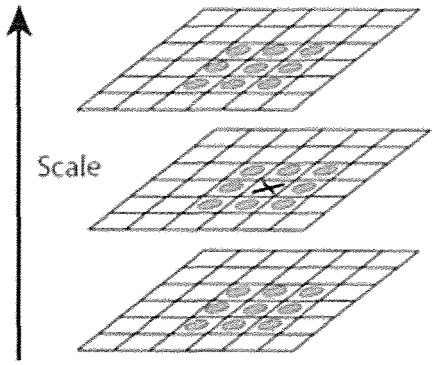


Figure 9: Maxima and Minima of the DoG images. Image from [18].

  The current pixel (marked with an x in the image) is defined to be a keypoint only if it is a minimum or a maximum amongst all of its neighbors. The cost of this check is quite low since most candidate points are rejected after the first few checks. These keypoints are the approximate maxima and minima. The approximation is due to the fact that maxima or minima never lie exactly on a particular pixel, but somewhere in between. Since one cannot simply access data in-between pixels, the sub-pixel must be located mathematically. Sub-pixel values are generated using the Taylor expansion.

  The next step for this stage is the rejection of some of the detected keypoints, either because they don't have enough contrast or lie on an edge. Such keypoints are not useful as features. Edge features are removed using an approach similar to the Harris Corner Detector while low contrast features are identified by checking their intensities.

- **Orientation assignment:**
  Having stable, scale-invariant keypoints, one or more orientations are assigned to each keypoint according to the gradient directions of the local image. All future operations are executed on image data which has been transformed with respect to the assigned orientation, scale, and location for each feature, therefore providing invariance to these transformations.

  Basically this stage involves the collection of the gradient directions and magnitudes of sample points in an area around each keypoint, and the most prominent orientation is

assigned to the keypoint. This orientation enables rotation invariance. An orientation histogram having 36 bins (thus covering the 360 degree range of orientations) is constructed from the collected data. Peaks in this histogram indicate the major directions of local gradients. The highest peak is identified and if there is any other local peak that is within 80% of the highest peak, it is used to create another keypoint with the respective orientation. Therefore, for keypoints having multiple peaks with similar magnitudes, a new keypoint is created for each major orientation. This results in a number of keypoints at the same location and scale but with different orientations. This only happens to about 15% of the points, but the latter contribute significantly to the stability of the matching process.

- **Keypoint descriptor:**
  The local image gradients are measured at the given scale in the area around each keypoint. These are converted into a representation which allows for substantial levels of local shape distortion and change in illumination.

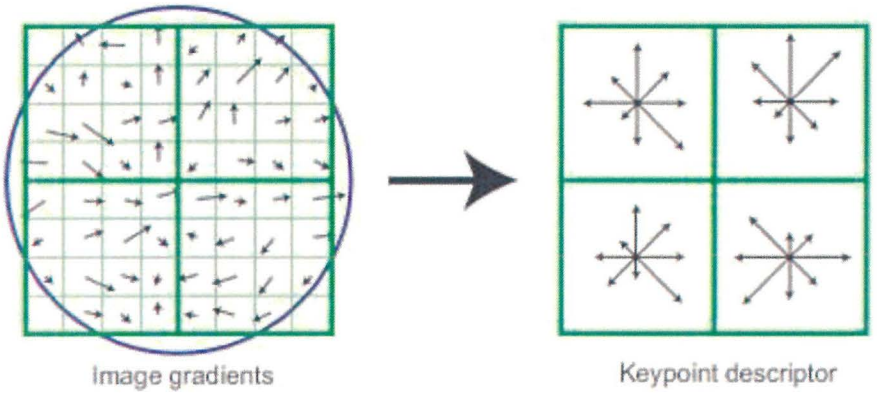


Figure 10: Keypoint descriptor. Image from [18].

Figure 10 depicts the computation of a keypoint descriptor. First, the gradients and magnitudes extracted from the image in the previous stage are weighted by a Gaussian Window shown by the superimposed circle. The purpose of the Gaussian Window is "to avoid sudden changes in the descriptor with small changes in the position of the window, and to give less emphasis to gradients that are far from the center of the descriptor" [18].

This data is then represented by orientation histograms, with a summary of the contents in 4x4 sub-regions, as shown on the right hand side of the image. The length of each arrow corresponds to the magnitude of the histogram entry in question. It is important to note that the figure shows a 2x2 descriptor array computed from an 8x8 set of samples; however the experiments led in [18] use 4x4 descriptors computed from a 16x16 sample array, resulting in 128 values for each keypoint descriptor.

To reduce the illumination dependence, numbers (from the 128 values) greater than 0.2 are given a threshold to be no larger than 0.2, and then the vector is normalized to unit length. The 0.2 value was "determined experimentally using images containing differing illuminations for the same 3D objects" [18]. To reduce rotation dependence, the keypoints'

rotations are subtracted from each orientation. Therefore gradient orientations are relative to the keypoints' orientations.

This approach matches new images by comparing each feature from the image in question with a database of SIFT features, and finds matching features based on Euclidean distance of their feature vectors. Correct matches can be identified from the full set of matches by finding sub-sets of keypoints which agree on the object in question and its location, scale and orientation in a new image. The determination of clusters can be quickly done using an efficient hash table implementation of the generalized Hough transform.

Each group of three or more keypoints which agree on a given object and its respective pose is then subjected to further confirmation. A least-squared estimate is made for an affine approximation to the object pose, and any image features matching the latter pose are identified, and outliers rejected. Then, given the accuracy of fit and number of probable false matches, the probability that a particular set of features indicates the presence of an object is determined. Object matches which pass all the mentioned tests can be defined as correct with high probability.

2.4.2.2 PCA SIFT

In [24] Ke and Sukthankar examine and improve upon the local image descriptor used by SIFT. The aim is to create a descriptor for the area of pixels around a keypoint's local neighborhood which is "compact, highly distinctive...and yet robust to changes in illumination and camera viewpoint" [24]. Similar to SIFT, the descriptors of PCA-SIFT encode the important aspects of the image gradient in the keypoint's neighborhood. Yet, instead of using SIFT's smoothed weighted histograms, Principal Components Analysis (PCA) is applied to the normalized gradient patch. PCA enables Ke and Sukthankar to "linearly-project high-dimensional samples onto a low-dimensional feature space" [24]. The latter projection can be computed once and saved.

Practically, PCA-SIFT is identical to SIFT in the first three stages of the approach. The fourth and final stage, namely the keypoint descriptor stage, builds a representation for each keypoint based on a patch of pixels in its local neighborhood. The PCA-SIFT algorithm for this stage accepts the same data as the standard SIFT descriptor, namely; sub-pixel location, scale, and dominant orientations of the keypoint. As stated in [24], PCA is a standard technique implemented in various computer vision problems such as feature selection, object recognition and face recognition, and, even though PCA has some shortcomings, it is still popular due to its simplicity.

The first stage of PCA-SIFT requires the pre-computation of an eigenspace to express the gradient images of local patches. After computing the first three stages of the SIFT algorithm, a 41x41 patch is extracted at the given scale and centered over the keypoint in question. This is then rotated to align its prominent orientation to a canonical orientation. Similar to SIFT, for keypoints with more than one prominent orientation, a representation for each orientation is built. The input vector required by the PCA-SIFT algorithm is then created by concatenating the horizontal and vertical gradient maps for the extracted patch at the keypoint at hand. This results in a vector of 2x39x39 = 3042 elements. This vector is then normalized to unit magnitude. The normalization is done to minimize the impact of illumination changes. One

should note that the 41x41 patch does not cover all of the pixel value space, nor the smaller number of regions extracted from natural images. Rather, it consists of the small set of patches which passed through the first three stages of the SIFT algorithm, more precisely [24]:

- "It is centered on a local maximum in scale-space;
- It has been rotated so that (one if its) dominant gradient orientations is aligned to be vertical;
- It only contains information for the scale appropriate to the keypoint – i.e. the 41x41 patch may have been created from a much larger region from the original image."

The eigenspace is then built by applying PCA to the covariance matrix of the extracted vectors. The matrix, made up of the top n eigenvectors, is serialized to disk and used as the projection matrix for PCA-SIFT.

The aim of the next stage for PCA-SIFT is to identify the feature vector for a particular image patch. This vector is considerably smaller than the standard SIFT feature vector, yet, it can be used with the same matching algorithms. The feature vector is calculated by creating its 3042-element normalized image gradient vector, and projecting it into the feature space using the pre-computed eigenspace. Ke and Sukthankar empirically determined good values for the dimensionality of feature space, n.

As mentioned, Euclidean distance between two feature vectors is used to determine whether they belong to the same feature within different images. The use of a fine-tuned threshold for this distance enables one to find a trade-off between false positives and false negatives.

2.4.2.3 ASIFT

This method, proposed by Morel and Yu in [25], simulates all possible views of an image by varying the two camera axis orientation parameters, namely the latitude and longitude angles. The ASIFT method thus covers the two parameters left over by SIFT. This addition to the SIFT approach results in a mathematically proved, fully affine invariant method whose complexity is approximately twice that of SIFT.

ASIFT simulates the scale and changes of the camera axis orientation and normalizes the rotation and translation. In fact, more specifically, ASIFT simulates the two camera axis parameters, then SIFT is implemented to simulate the scale and normalize the rotation and translation. Morel and Yu introduce an invaluable parameter for the evaluation of the performance of affine recognition, namely the transition tilt. The latter parameter measures the degree of viewpoint change between different views of the same object.

Figure 11 depicts one of the first perspectively correct Renaissance paintings by Paolo Uccello. One can see how the perspective on the floor is strongly projective; the rectangular pavement of the room takes a trapezoidal form. Yet, each tile on the floor is nearly a parallelogram. This shows the "local tangency of perspective deformations to affine maps" [25]. In fact, by implementing an affine map, by the first order Taylor formula, any planar smooth deformation can be approximated around each point.

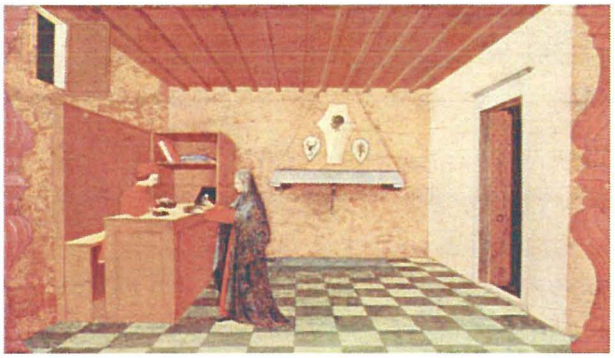


Figure 11: The global deformation of the ground is strongly projective (a rectangle becomes a trapezoid), but the local deformation is affine: each tile on the pavement is almost a parallelogram. Image from [24].

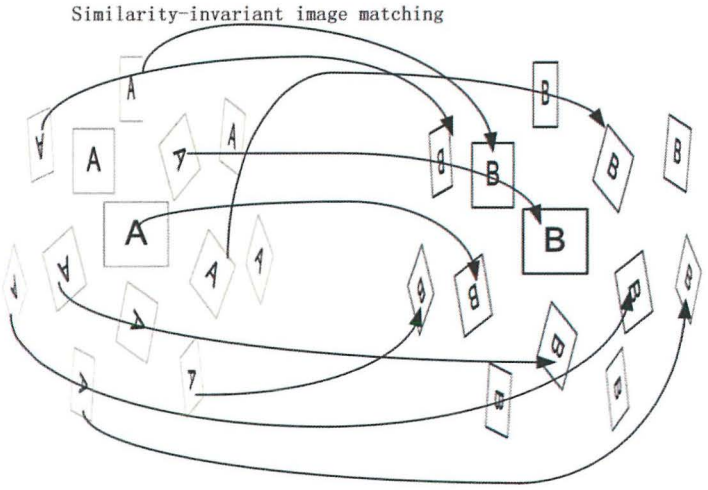


Figure 12: Overview of the ASIFT algorithm. Image from [24].

Figure 12 represents an overview of the ASIFT algorithm. The perfect square images, A and B, represent the two images to be compared. The simulated images, represented by the parallelograms, are the simulations of the changes brought about by a variation of the camera optical axis direction. The latter images are then compared using SIFT, which is rotation, scale, and translation invariant.

A key observation is that a tilt distortion, even though it is "irreversible due to its non-commutation with the blur, [it] can be compensated up to a scale change by digitally simulating a tilt of same amount in the orthogonal direction" [25]. ASIFT makes use of simulation instead of normalization methods, which suffer from the mentioned non-commutation. This enables ASIFT to be fully affine invariant. It is important to note that the simulation of the whole affine space with the proposed affine space sampling is not restrictive. Besides, the implementation of a two-resolution scheme reduces the ASIFT complexity to approximately twice that of SIFT.

The ASIFT algorithm can be summarized by the following stages:

- First, every image is transformed by a simulation of all possible affine distortions caused by "the change of camera optical axis orientation from a frontal position" [25]. These distortions are determined by the longitude and latitude parameters;
- The images undergo rotations and tilts for a finite and a small number of latitude and longitude angles. The sampling steps of the latter parameters ensure that the simulated images are kept close to any other possible view obtained by other values of latitude and longitude;
- SIFT is implemented to compare all the simulated images.

The acceleration with two resolutions involves applying ASIFT on low-resolution versions of a query and the search images. In case of a match, the affine transforms which resulted in matches in the low-resolution process are selected. These transforms are then simulated on the original query and search images. Finally, the generated images are compared using the SIFT algorithm.

2.4.2.4 SURF

In [26], the authors present SURF: a scale and rotation invariant interest point detector and descriptor. Their approach relies on integral images for image convolutions, implements a Hessian matrix-based measure for the detector, and a distribution-based descriptor, and simplifies the latter methods to the essential. More precisely, the detector uses a very basic approximation of the Hessian matrix. The approximation relies on integral images to reduce the computation time. Thus, the detector is known as a 'Fast-Hessian' detector. The descriptor, on the other hand, describes "a distribution of Haar-wavelet responses within the interest point neighborhood" [26]. Yet again, the integral images are exploited for speed. Apart from this, only 64 dimensions are used. This results in quicker computation and matching, while at the same time increasing robustness. The authors also introduce a new indexing step based on the sign of the Laplacian which, apart from increasing the robustness of the descriptor, also increases the matching speed.

The following stages describe the Fast-Hessian Detector:

- **The Hessian**

  The SURF detector is based on the Hessian matrix since it has good performance in computation time and accuracy. To be more precise, blob-like forms are identified at locations where the determinant is maximum. Yet, instead of using different measures for the selection of the location and the scale, the determinant of the Hessian is used for both. The latter value, known as the discriminant, is used as a means of classification for the maxima and minima of the function. This discriminant is a product of eigenvalues of the Hessian, thus, points can be classified according to the sign of the result. "If the determinant is negative then the eigenvalues have different signs and hence the point is not a local extremum; if it is positive then either both eigenvalues are positive or both are negative and in either case the point is classified as an extremum" [27]. The use of Gaussian in the described process allows the varying of the amount of smoothing during the convolution stage. This means that the determinant is calculated at various scales. However, the authors of [26] explore a simpler alternative to the Gaussian filters, and

extend Lowe's LoG approximations by using box filters. The latter, shown in ***Figure 13***, approximate second order Gaussian derivatives and, using integral images, can be evaluated very fast independently of size.

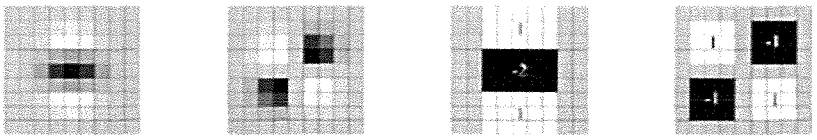


Figure 13: Left to right - The (discretised and cropped) Gaussian second order partial derivatives in y-direction and xy-direction, and the SURF approximations using box filters. The grey regions are equal to zero. Image from [26].

- **Constructing the Scale-Space**

Interest points require to be identified within different scales, since the search of matches often requires their comparison in images where they are seen within different scales. Scale spaces, usually implemented as an image pyramid, are repeatedly smoothed with a Guassian and then scaled to a smaller size in order to achieve a higher level of the pyramid. Since SURF uses box filters and integral images, there is no need to iteratively apply the same filter to the output of a previously filtered layer. Instead, box filters of any given size can be applied directly on the original image. The use of integral images reduces the computation time drastically.

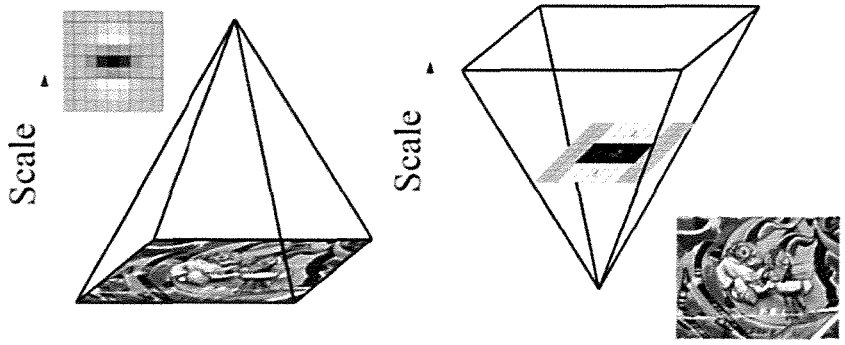


Figure 14: Filter Pyramid - The left hand side portrays the traditional approach to constructing a scale space while the right hand side shows the SURF approach. Image from [27].

The difference between the traditional approach and the approach used in SURF to constructing a scale-space can be seen in ***Figure 14***. In the traditional approach, the image size is varied and the Gaussian filter is applied to smooth subsequent layers. The SURF approach, on the other hand, leaves the original image unchanged and varies only the filter size. For a more detailed description of scale space refer to Section 2.4.2.1.

- **Accurate Interest Point Localization**

The localization of scale and rotation invariant interest points in an image starts with the thresholding of the responses in such a way that all values under a pre-determined

threshold are discarded. The latter threshold can be adapted according to the requirements in question; a higher threshold value leaves only the strongest interest points and vice versa.

Next, each pixel in the constructed scale space is compared to its neighbors in the scale in question, and the scales above and below it, as can be seen in ***Figure 9***. The latter process is done to find a set of candidate interest points. This stage results in a set of interest points "with minimum strength determined by the threshold value and which are also local maxima/minima in the scale-space" [27].

The final process of this stage involves the interpolation of data near the interest points to find the location in "both space and scale to sub-pixel accuracy" [27]. This is done by expressing the determinant of the Hessian function as a Taylor expansion centered at the detected location. This process leaves only the most stable interest points.

The SURF interest point descriptor describes the distribution of the pixel intensities within a scale dependent region of keypoints identified by the Fast-Hessian. The approach used by SURF exploits integral images for speed, together with filters known as Haar wavelets, to increase robustness and decrease computation time. Haar wavelets, as can be seen in ***Figure 15***, are used to find gradients in the x and y directions. When used together with integral images, each wavelet requires only six operations to be computed.

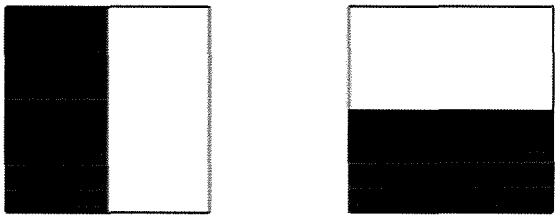


Figure 15: Haar wavelets - The left filter computes the response in the x-direction while the right filter computes the y-direction. Weights are 1 for black regions and -1 for white regions. Image from [27].

The first process for the extraction of the descriptor involves the assignment of an orientation to each interest point. Then, a scale-dependent window is constructed in which a 64-dimensional vector is extracted. It is vital that measurements respective to the detected scale are used in all calculations for the descriptor to ensure scale-invariant results.

- **Orientation Assignment**

Rotation invariance is achieved by assigning a reproducible orientation to each detected interest point. It is important that such an orientation is found to be repeatable under varying conditions, since extraction of the descriptor components is computed relative to this direction. Orientation assignment is done by calculating Haar wavelet responses for a set of pixels within a given radius of the detected point, according to the scale at which the point was detected.

The responses are then weighted with a Gaussian centered at the interest point in question. This Gaussian is also dependent on the scale of the latter point. After they are weighted, the results are represented as points in vector space. The most prominent orientation is chosen by rotating a circle segment around the origin, and summing up the x and y-responses within the segment to form a new vector. The longest vector is assigned as the interest point orientation.

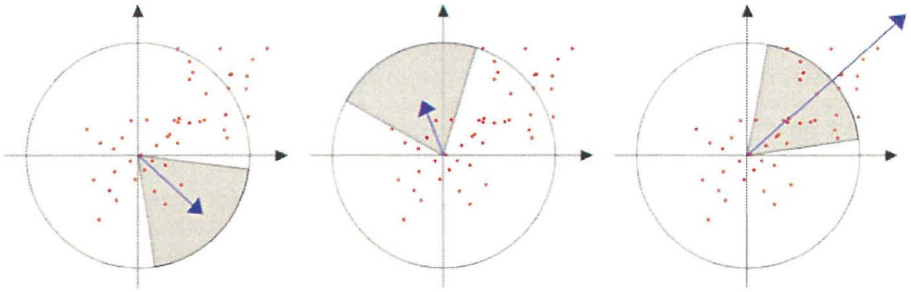


Figure 16: Orientation assignment. Image from [27].

- **Descriptor components**

Firstly, a square window containing the pixels which will form values in the descriptor vector is constructed around the interest point in question. This window is oriented along the dominant orientation found in the previous stage, in such a way that all subsequent calculations are relative to this orientation.

Figure 17: Descriptor Windows - Each window is 20 times the size of the scale of the detected point and is oriented along the dominant direction shown in green. Image from [27].

The descriptor windows are divided into 4x4 regular sub-regions. For each sub-region, Haar wavelets are calculated for 25 regularly distributed sample points, resulting in four values. This means that the descriptor vector is of length 4x4x4 = 64. The resulting SURF descriptor is rotation, scale and brightness invariant, and, after reduction to unit length, even contrast-invariant.

For the matching stage, for faster indexing, the sign of the Laplacian for the underlying interest point is included in the descriptor. The sign of the Laplacian enables the distinguishing of bright blobs on dark backgrounds from the opposite situation. Since this feature is computed during the detection stage, it is available at no additional computational cost. For the matching process, only features with the same type of contrast are compared, thus, this minimal information enables faster matching and a slight increase in performance.

2.5 Feature Matching

The use of features in object recognition requires a matching approach which compares individual features to a database of features and discards false positive matching feature pairs. Many features will be incorrectly matched due to ambiguous features, or features arising from clutter. As described in [18], if clusters of at least 3 features are identified which agree on object and pose, then the feature has a high probability of being a correct match.

The nearest neighbor approach involves identifying the best candidate by finding the nearest neighbor of a keypoint from a database of features. The nearest neighbor can be defined as the keypoint with the minimum Euclidean distance for the feature descriptor. Simply finding the nearest neighbor however, will not give the best results, as many features extracted from the image in question can be derived from clutter, thus there will be no matching features in the database. The case might also be that the features were not detected in the training images. This problem requires a means to reject features which do not have a good match in the database. A global threshold on the distance to the nearest neighbor would not be useful here, since some descriptors are more discriminative then others. An effective solution, as proposed by [18], would be to set a threshold on the ratio between the distance of the first nearest neighbor to the distance of the second nearest neighbor. If the database contains multiple training images of the same object, then the second nearest neighbor is taken to be the first nearest neighbor from a second unrelated image. The threshold on the ratio between distances is effective since correct matches require having the closest nearest neighbor significantly closer than the second-closest nearest neighbor. Thus, false matches will be distinguished by a large ratio between the first and second nearest neighbors. ***Figure 21*** depicts the probability of a correct match by using the nearest neighbor ratio matching.

The proposed solution has however one major setback; nearest neighbor search times depend exponentially on the dimension of the space. This is coined as "the curse of dimensionality". Due to this setback, many algorithms cannot be more efficient than exhaustive search in the identification of exact nearest neighbors of points in high dimensional spaces. Taking the example of SIFT's 128 dimensional feature vector, the kd tree algorithm [28] would provide no speedup over exhaustive search for more than about 10 dimensional spaces. Following are some of the most feasible solutions.

2.5.1 Best-Bin-First matching

In [29], Lowe and Beis propose a variant of the kd-tree search algorithm which makes indexing in higher dimensional spaces practical. The proposed search approach, coined Best Bin First (BBF) search, is an approximate algorithm which finds the nearest neighbor to a query point for a large number of queries, while finding a very close neighbor for the remaining cases. Even though the developing of such an algorithm was done with the application of shape indexing in

mind, this search approach can be used efficiently for closest point matching in appearance-based recognition [30].

The standard kd-tree begins with a complete set of N points in R^k. The data space is split on the dimension i in which the data shows the greatest variance. A cut is performed at the median value m of the data in dimension i in such a way that both sides of the cut have an equivalent number of points. An internal node is created to store i and m, and the process is repeated with both halves of the data. This process results in a balanced binary tree with depth $d = [\log_2 N]$. The leaves of a kd-tree form a complete section of the data space. An interesting property one should note is that bins are smaller in higher-density areas and larger in lower density regions. Thus, with high probability, the nearest neighbor to a query should lie in the bin where the query falls, or in a bin adjacent to it.

To find the nearest neighbor to a query point q, a "backtracking, branch-and-bound search" [29] is implemented. This involves traversing the tree to find the bin containing the query point. This step requires only d scalar comparisons, and gives a good approximation to the nearest neighbor. Then, the backtracking stage requires the pruning of whole branches of the tree if the region of space they represent is further from the query point than the distance from q to the closest neighbor yet seen. The described process can be very effective in low-dimensional spaces, but, since higher dimensions result in many more bins adjacent to the central one that must be examined, performance degrades rapidly. It is important to note that a great deal of the search time is spent examining bins in which only a small fraction of their volume could provide the nearest neighbor [29]. This means that if an approximation to the nearest neighbor is sufficient to the problem at hand, then sustained search can be avoided by posing a limit on the number of leaf nodes to be examined (E_{max}), and settling for the best neighbor found up to that point.

Even though the mentioned modification extends by a small amount the domain of kd- tress for fast nearest neighbor, the backtracking step described earlier is still inefficient. This is due to the order of examining leaf nodes, which is done according to the tree structure and depends only on the stored points. Also, the order does not consider the position of the query point. An optimization could be looking in bins in order of increasing distance from the query point. The distance to a bin is defined as "the minimum distance between q and any point on the bin boundary" [29].

The BBF search strategy proposed provides a drastic improvement in nearest neighbor search for moderate dimensionality (e.g. 8-15) and, for fairly small values of E_{max}, this approach identifies the exact nearest neighbors a large percentage of the time, and gives a very close neighbor the remaining times.

2.5.2 *RANDOMIZED KD-TREE ALGORITHM*

Silpa-Anan and Hartley in [31] propose an improved version of the kd-tree algorithm where multiple randomized kd-trees are generated. As opposed to the original kd-tree algorithm, the randomized trees are generated by choosing the split dimension randomly from the first D dimensions on which data has the greatest variance.

When searching the trees, a single priority queue is maintained across all the randomized trees with the aim of ordering the search by increasing distance to each bin boundary. The degree of approximation is determined by examining a fixed number of leaf nodes. When this threshold is reached, the search is stopped and the best candidates are returned.

According to experiments led by Lowe and Muja in [32], performance of randomized kd-trees improves with the number of generated trees up to a certain point. Increasing the number of random trees beyond this point leads to static or decreasing performance.

2.5.3 *HIERARCHICAL K-MEANS TREE*

Lowe and Muja in [32] present an algorithm which applies priority search on hierarchical k-means trees, as opposed to branch-and-bound approaches which search in depth-first order. The hierarchical k-means tree is built by separating the data points at each level into K distinct areas using a k-means clustering. The same method is applied recursively to the points in each area. This recursion is stopped when the number of points in an area is smaller than K.

Lowe and Muja developed an algorithm which explores the hierarchical k-means tree in a BBF manner. This algorithm starts by performing a single traversal through the tree and appends to a priority queue all the unexplored branches in each node along the traversed path. Then, the branch which has the closest center to the query point is extracted from the priority queue, and the tree traversal is restarted from that branch. As it traverses the branches in turn, the algorithm keeps appending to the priority queue the unexplored branches along the path. The degree of approximation is specified similarly as for the randomized kd-trees; by stopping the search after a specified number of leaf nodes (dataset points) have been examined.

The hierarchical k-means tree performs best with most datasets, however, a disadvantage of this algorithm is that it often has a higher tree-build time than the randomized kd-trees. This build-time can be reduced significantly by executing a small number of iterations in the k-means clustering stage instead of running it until convergence.

2.6 FEATURE-BASED ALGORITHMS EVALUATION

In the papers presenting SIFT [18], PCA-SIFT [24] and SURF [26], the authors perform an evaluation on the proposed algorithms. These evaluations are done with the purpose of determining various properties of the detectors and descriptors of each algorithm.

2.6.1 *SIFT*

In [18] Lowe conducts various experiments to evaluate the SIFT descriptor. One of which is the varying of two parameters used to vary the complexity of the descriptor, namely the number of orientations, r, in the histograms and the width, n, of the n x n array of orientation histograms. The resulting descriptor vector can be described as rn^2. The more complex a descriptor is, the more it discriminates when using a large database of features, however it is more susceptible to shape distortions and occlusion. ***Figure 18*** depicts this experiment. The graph was generated using images with a viewpoint transformation where a planar surface is tilted by 50 degrees away from the viewer and 4% noise is added. Such images present the descriptor with the most extreme data for reliable matching. The graph shows the percentage of keypoints which find a correct match to a single nearest neighbor from a database of 40,000 features. Results show that

a single orientation histogram (n = 1) has a very poor discrimination. The best n-value is found to be 4, with 8 orientations. More orientations or a larger descriptor would end up making the descriptor more sensitive to distortion.

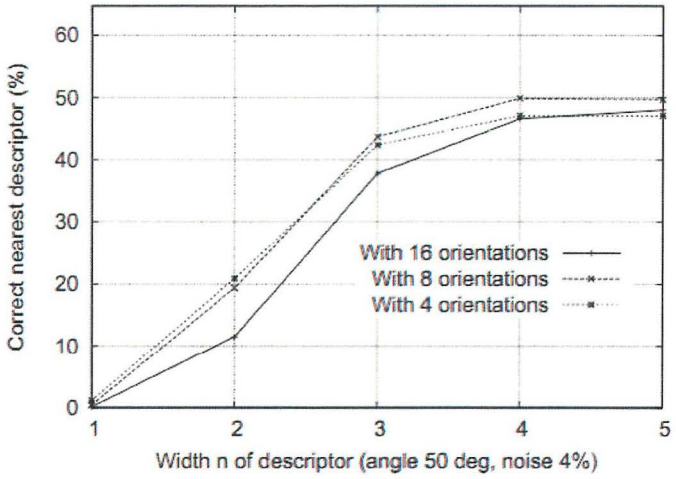


Figure 18: The percentage of keypoints giving the correct match to a database of 40,000 keypoints as a function of width of the n x n keypoint descriptor and the number of orientations in each histogram. Image from [18].

Figure 19 shows the experiment led to evaluate the sensitivity of the descriptor to affine change. The graph shows "the reliability of keypoint location and scale selection, orientation assignment, and nearest neighbor matching to a database as a function of rotation in depth of a plane away from a viewer" [18]. Results show that there is a reduced percentage of repeatability at each stage of computation with increasing amounts of affine distortion. Yet, up to a 50 degree change in viewpoint, the final matching accuracy remains above 50%.

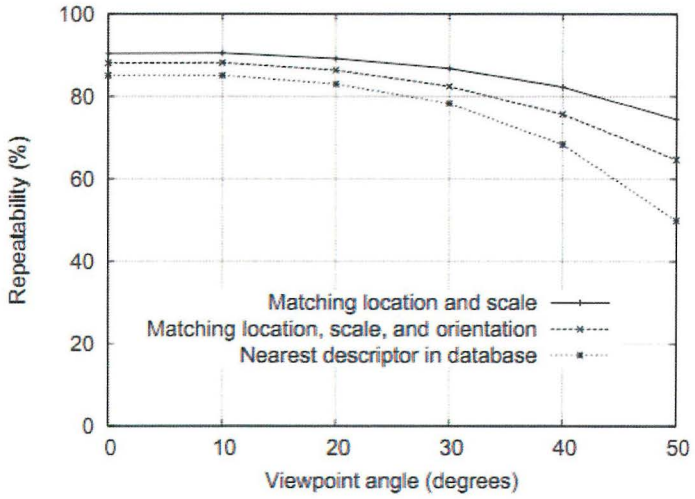


Figure 19: The stability of detection for keypoint location, orientation, and final matching to a database as a function of affine distortion. The degree of affine distortion is expressed in terms of the equivalent viewpoint rotation in depth for a planar surface. Image from [18].

Lowe conducted an evaluation to determine the varying in the reliability of matching as a function of the number of features in the database of keypoints. The graph in ***Figure 20*** was generated using a database of 112 images with a viewpoint depth rotation of 30 degrees and 2% image noise, along with normal random image rotation and scale change. The leftmost point on the dashed line shows the matching against features from a single image while the rightmost point is the matching against all features in the database. The reason that the solid line is flat is that the testing of the number of keypoints which were identified at the respective matching location and orientation in the transformed image was run over the full database for each value. The results show that the larger the database, the less reliable the matching is, however, correct matches will still be found even with very large databases. The small gap between the two lines indicates that a failure in finding a match is more probable to be due to problems with the initial feature localization and orientation assignment, than due to the feature distinctiveness. This is true even in large databases.

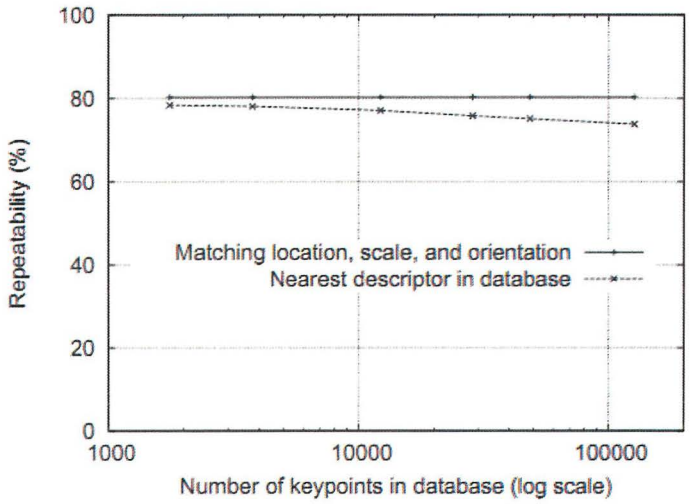


Figure 20: The dashed line shows the percentage of keypoints correctly matched to a database as a function of database size (using logarithmic scale). The solid line shows the percentage of keypoints assigned the correct location, scale, and orientation. Image from [18].

Images with random scale and orientation change, a depth rotation of 30 degrees, and addition of 2% image noise were used to evaluate the probability of a correct match against a database of 40,000 features. ***Figure 21*** shows the results. All matches in which the distance ratio between the closest neighbor and the second closest neighbor is larger than 0.8 are discarded. This rejects 90% of false matches while it discards only 5% of the correct matches. This approach is known as nearest neighbor distance ratio matching.

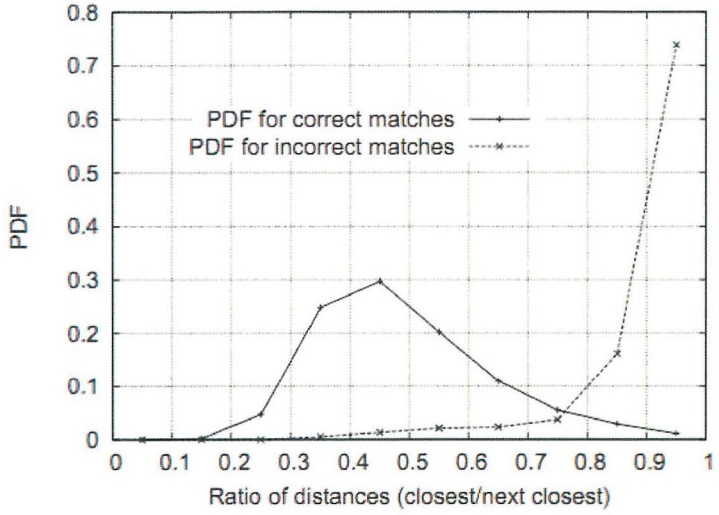


Figure 21: The probability that a match is correct can be determined by taking the ratio of distance from the closest neighbor to the distance of the second closest. The solid line shows the Probability Density Functions for correct matches, while the dotted line is for matches that were incorrect. Image from [18].

2.6.2 PCA-SIFT

Apart from experimenting with the relationship between the dimensionality of PCA-SIFT and the matching accuracy, the authors of [24] run three main types of experiments with the aim of evaluating the difference between standard SIFT representation and PCA-SIFT.

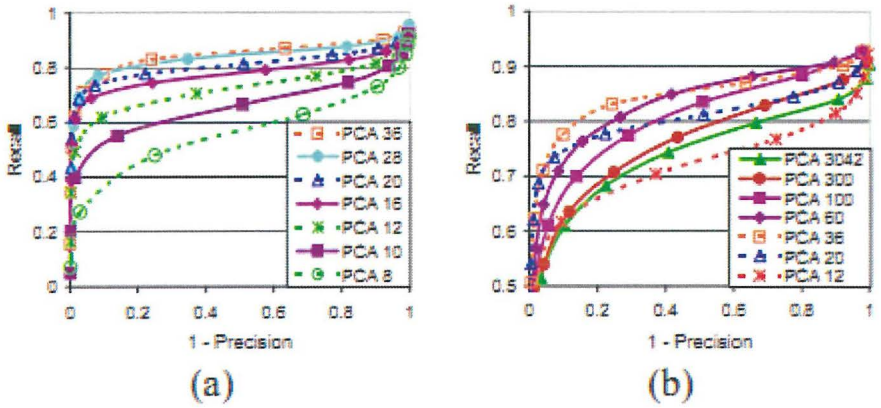


Figure 22: PCA-SIFT performance as PCA dimension (n) is varied. Image from [24].

Figure 22 shows the relationship between the matching accuracy of PCA-SIFT and the dimensionality of the feature space. As can be seen, 36 is the best value for n, with higher dimensions actually resulting in a decline of matching accuracy. It is important to note that a larger dimensionality requires a longer running time.

The first experiment type was aimed to examine the robustness of both descriptors to artificially created effects; addition of noise, changes in illumination and the application of image transformations. The graphs showing the results are generated as follows:

- The keypoints of all images in the dataset are extracted (using the initial stages of SIFT);
- All pairs of keypoints from different images are examined – if the Euclidean distance between feature vectors for the pair of keypoints in question is beneath a given threshold, the pair is defined a match. If two keypoints correspond to the same physical location, the match is termed a correct-positive, otherwise, it is a false-positive.

The evaluation metrics used are recall vs. 1-precision:

$$recall = \frac{number\ of\ correct\ positives}{total\ number\ of\ positives}$$

$$1 - precision = \frac{number\ of\ false\ positives}{total\ number\ of\ matches\ (correct\ or\ false)}$$

Equation 3: Evaluation metrics equations

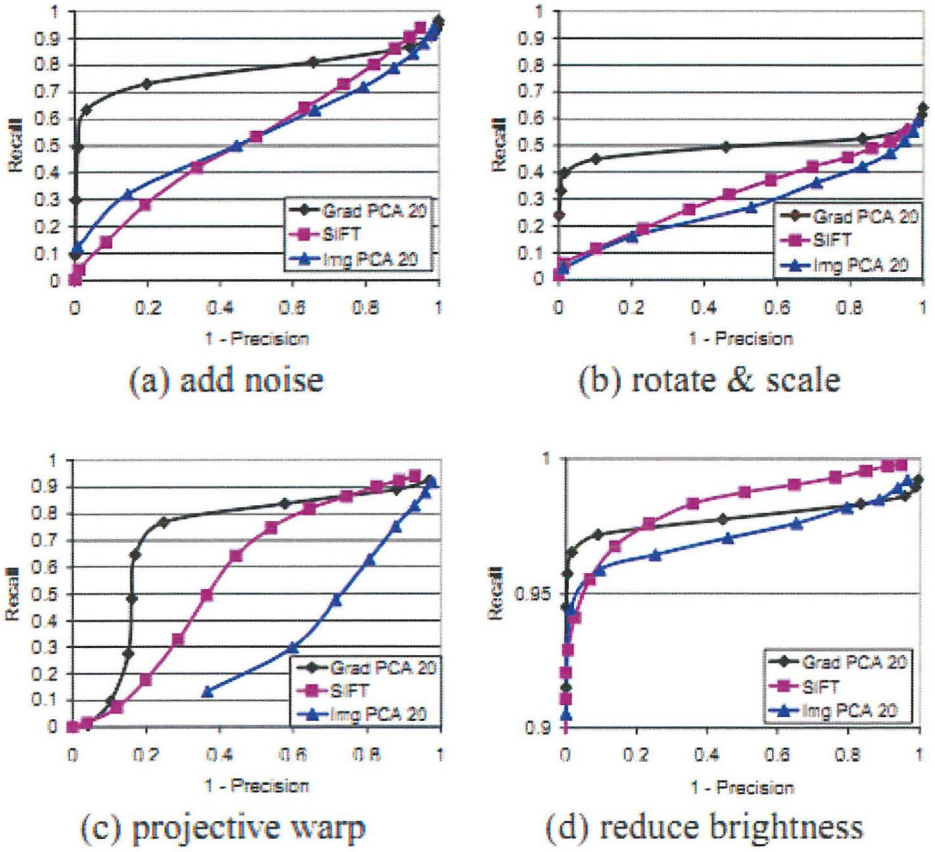


Figure 23: SIFT vs. PCA-SIFT on a matching task where the images are deformed or corrupted under controlled conditions. (a) Target images were corrupted by Gaussian noise. (b) Target images were rotated by 45 degrees and scaled by 50%. (c) Target images were projectively warped to simulate a viewpoint change of 30 degrees. (d) Target image intensities were scaled by 50%. Image from [24].

Figure 23 depicts the results for the first set of matching experiments. ***Figure 23***(a) shows that PCA-SIFT is much better at handling noise in images for nearly all values of 1 – precision. SIFT only outperforms PCA-SIFT when extremely high false positive rates can be tolerated. ***Figure 23***(b) shows the results where target images were rotated by 45 degrees and scaled by 50%. ***Figure 23***(c) on the other hand shows matches after target images were distorted by a perspective transformation equivalent to a 30 degrees out-of-plane rotation. Finally, ***Figure 23***(d) shows that all descriptors are capable of handling illumination changes, with a recall of more than 95%.

The second experiment type was run using real images taken from varying viewpoints. ***Figure 24*** shows the results: although the absolute recall is quite low, PCA-SIFT definitely gives the best results.

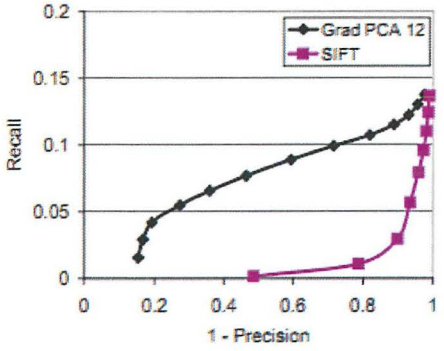


Figure 24: SIFT vs. PCA-SIFT using real images taken from varying viewpoints. Image from [24].

The last experiment involved the integration of SIFT and PCA-SIFT in an image retrieval application. Given two images, the corresponding feature vectors were extracted. Each feature vector in one image was compared against all feature vectors in the other image in turn, and the number of features falling within a threshold distance were counted. In short, the experiment required the use of a small dataset of 3 images each of 10 household items from different viewpoints. 2 images of each object were added to the database while the remaining images were used as query images. The algorithm in question was awarded points according to the result of the image retrieval process: if the top 2 images corresponding to the query were returned the algorithm was awarded 2 points, if 1 image was returned, 1 point is awarded, otherwise 0 points were awarded. The resultant scores were divided by 60 (the number of correct matches). This experiment resulted in the SIFT obtaining 43% and PCA-SIFT obtaining 68% correctly retrieved images. Results also showed that the localization of features in images took comparable times for both SIFT and PCA-SIFT, but PCA-SIFT is much faster in the matching stage.

To study the reason behind SIFT's matching success, namely the careful design of its feature representation to be robust to localization error, another experiment was led where registration error was introduced after the localization stage. When error was introduced into the orientation assignment phase (1 and 3 pixel errors at the appropriate scale in a random direction with respect to the prevailing orientation), or in the scale estimation, PCA-SIFT's accuracy declines. This confirms that SIFT is better suited at handling such errors.

2.6.3 ASIFT

Experiments presented in [25] include extensive tests using the Mikolajczyk database [33] and a thorough evaluation of various methods' invariance to absolute and transition tilts. Here we focus on the results obtained by SIFT and ASIFT, since the other methods are not relevant in this context.

The Mikolajczyk database was used to evaluate the robustness to four types of distortions, namely blur, similarity, change in viewpoint and JPEG compression. The results show that ASIFT performs best in these tests. This is shown in ***Figure 25***.

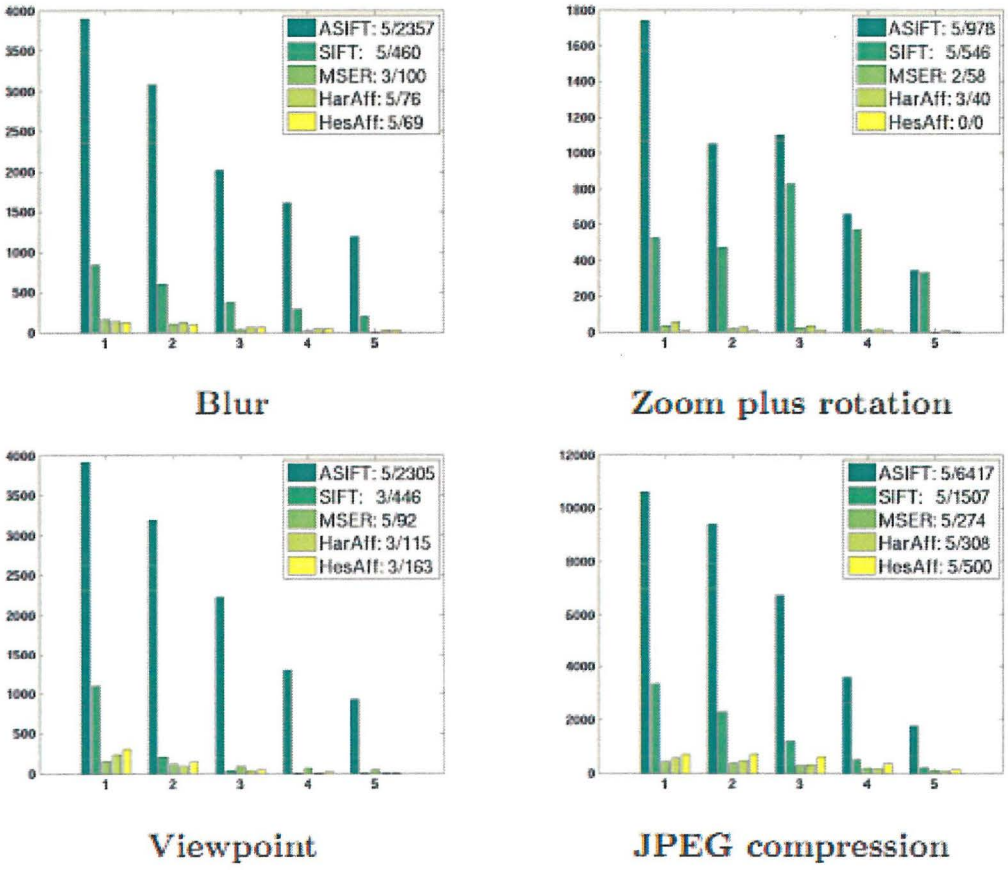


Figure 25: Number of correct matches achieved by ASIFT and SIFT under four types of distortions. On the top-right corner of each graph *m/n* gives for each method the number of image pairs *m* on which more than 20 correct matches were detected, and the average number of matches *n* over these *m* pairs. Image from [25].

Figure 26 (a) shows the settings adopted for the absolute tilt test. A planar object was photographed with an optical zoom varying from x1 to x10 and with viewpoint angles between the camera axis and the normal to the object varying from 0 degrees (frontal view) to 80 degrees. ***Figure 28*** depicts the results; for zoom x1, the number of sift matches drops considerably when the angle is larger than 65 degrees and fails completely when the angle is larger than 75 degrees. ASIFT, on the other hand, functions perfectly till 80 degrees. For images

taken at zoom x10, SIFT performance is possible only with angles smaller than 45 degrees. Again, ASIFT works perfectly until 80 degrees.

For the transition tilt test a planar object is photographed as depicted in ***Figure 26*** (b) to obtain two sets of images. For each image set, the camera, set with a fixed latitude angle equivalent to t=2 and t=4, is circled around with the longitude angle growing from 0 to 90 degrees. The optimal zoom and the focus distance were set to x4. For absolute tilt of 2, the SIFT performance drops considerably when the transition tilt goes from 1.3 to 1.7. Under an absolute tilt of 4, SIFT struggles at a 1.9 transition tilt, and fails completely when the tilt gets bigger. ASIFT works perfectly for transition tilts up to 16, with the experimental limit transition tilt going easily up to 36. The results are depicted in ***Figure 29***.

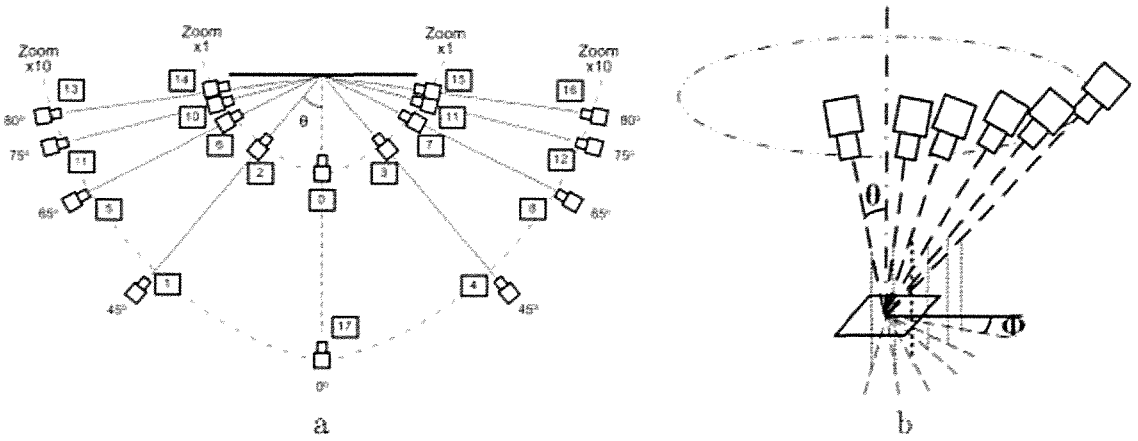


Figure 26: The settings adopted for systematic comparison. (a) Absolute tilt test (b) Transition tilt. Image from [25].

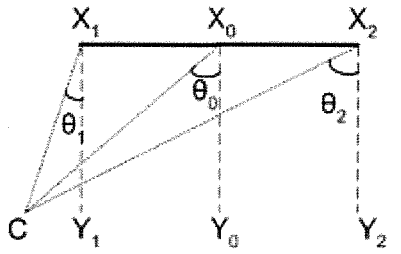


Figure 27: When the camera focus distance is small, the absolute tilt of a plane object can vary considerably in the same image due to the strong perspective effect. Image from [25].

Z × 1					
θ/t	SIFT	HarAff	HesAff	MSER	ASIFT
−80°/5.8	1	16	1	4	110
−75°/3.9	24	36	7	3	281
−65°/2.3	117	43	36	5	483
−45°/1.4	245	83	51	13	559
45°/1.4	195	86	26	12	428
65°/2.4	92	58	32	11	444
75°/3.9	15	3	1	5	202
80°/5.8	2	6	6	5	204
Z × 10					
θ/t	SIFT	HarAff	HesAff	MSER	ASIFT
−80°/5.8	1	1	0	2	116
−75°/3.9	0	3	0	6	265
−65°/2.3	10	22	16	10	542
−45°/1.4	182	68	45	19	722
45°/1.4	171	54	26	15	707
65°/2.4	5	12	5	6	468
75°/3.9	2	1	0	4	152
80°/5.8	3	0	0	2	110

Figure 28: Absolute tilt invariance comparison – number of correct matches for viewpoint angles between 45 degrees and 80 degrees. Top: images taken with zoom x1 Bottom: images taken with zoom x 10. The latitude angles and the absolute tilts are listed in the left column. For the x1 zoom, strong perspective effect is present and the tilts shown are average values. Image from [25].

$t_1 = t_2 = 2$					
ϕ_2/τ	SIFT	HarAff	HesAff	MSER	ASIFT
10°/1.3	408	233	176	124	1213
20°/1.7	49	75	84	122	1173
30°/2.1	5	24	32	103	1048
40°/2.5	3	13	29	88	809
50°/3.0	3	1	3	87	745
60°/3.4	2	0	1	62	744
70°/3.7	0	0	0	51	557
80°/3.9	0	0	0	51	589
90°/4.0	0	0	1	56	615
$t_1 = t_2 = 4$					
ϕ_2/τ	SIFT	HarAff	HesAff	MSER	ASIFT
10°/1.9	22	32	14	49	1054
20°/3.3	4	5	1	39	842
30°/5.3	3	2	1	32	564
40°/7.7	0	0	0	28	351
50°/10.2	0	0	0	19	293
60°/12.4	1	0	0	17	145
70°/14.3	0	0	0	13	90
80°/15.6	0	0	0	12	106
90°/16.0	0	0	0	9	88

Figure 29: Transition tilt invariance comparison. Number of correct matches of ASIFT and sift for viewpoint angles between 50 AND 80 degrees. Image from [25].

2.6.4 SURF

The authors of [26] evaluate SURF using the images and testing software provided by Mikolajczyk [33]. The SURF descriptor is compared to SIFT, PCA-SIFT and GLOH; a SIFT-like descriptor. SURF resulted the best descriptor for almost all comparisons. ***Figure 30*** depicts matching results obtained with two different matching techniques, namely a similarity-threshold-based strategy and a nearest-neighbor-ratio matching strategy. Tests were done on the 'Graffiti' images, with a view change of 30 degrees. ***Figure 32*** shows recall vs. (1-precision) graphs using various Mikolajczyk image sets using similarity-threshold-based matching.

As can be seen from the diagrams, the SURF descriptor outperforms the other descriptors with sometimes more than 10% improvement in the recall for the same level of precision. SURF-128 corresponds to the extended SURF descriptor, which shows slightly better results than standard SURF. SURF-128 however has a longer matching process. This is shown in ***Figure 31***.

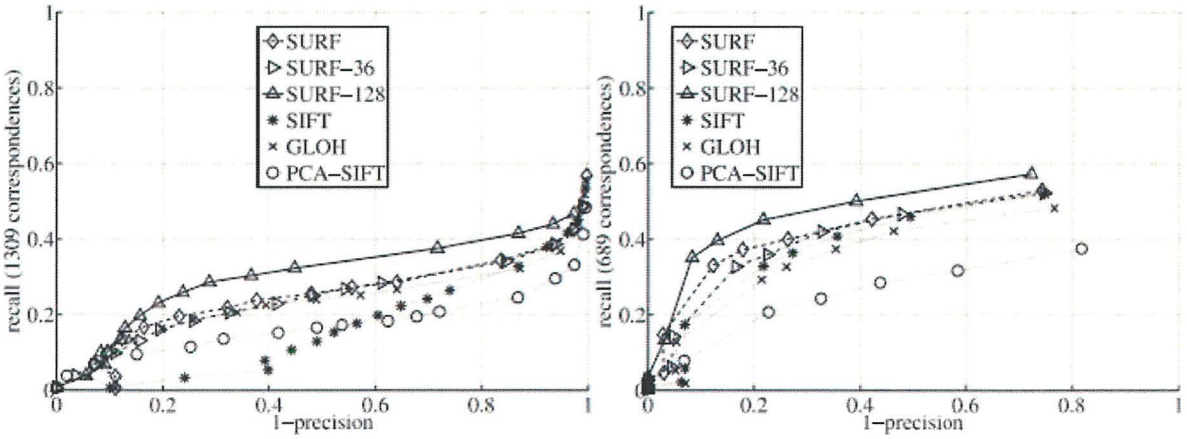


Figure 30: Recall vs. 1-precision graph for different binning methods and two different matching strategies. Left: Similarity-threshold-based matching strategy. Right: Nearest-neighbor-ratio matching strategy. Image from [26].

	U-SURF	SURF	SURF-128	SIFT
time (ms):	255	354	391	1036

Figure 31: Computation times for the joint detector-descriptor implementations. The thresholds are adapted in order to detect the same number of interest points for all methods. Image from [26].

The SURF approach was also tested in a practical application with the aim of recognizing art objects in a museum. A database of 216 images depicting 22 objects was built using images taken under various conditions. The conditions included extreme lighting changes, images of objects within reflecting glass cabinets, changes in viewpoint, zoom, various camera qualities, etc. The resolution used was 320x240. Using the nearest neighbor matching strategy, the results ranked with the following recognition rate:

- SURF-128 – 85.7%

- U-SURF (Upright SURF) – 83.8%
- SURF – 82.6%
- GLOH – 78.3%
- SIFT – 78.1%
- PCA-SIFT – 72.3%

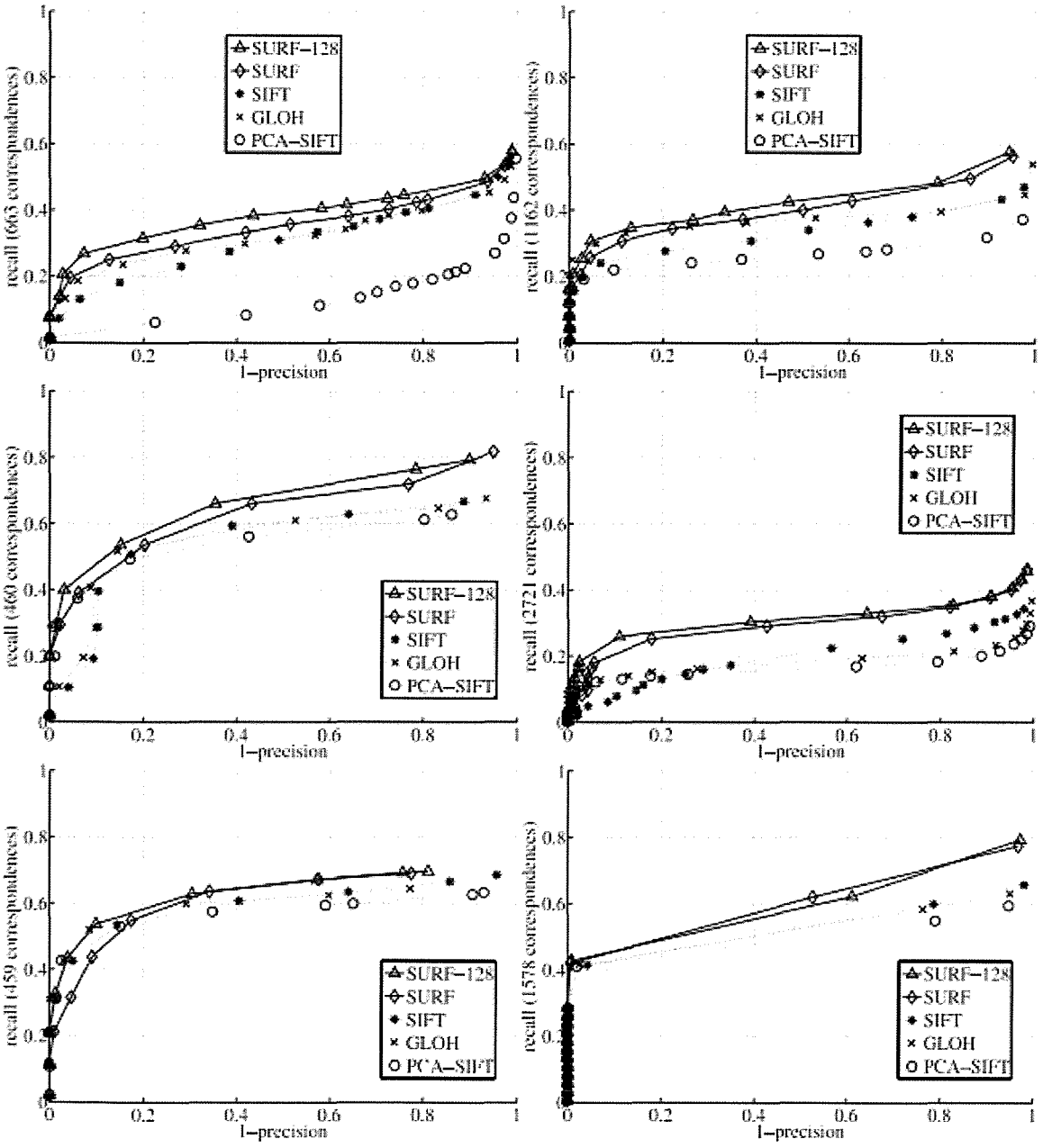


Figure 32 : Recall vs. 1-precision graphs for, from left to right and top to bottom, Viewpoint change of 50 degrees (wall), scale factor of 2 (Boat), image blur (Bikes), image blur (Trees), brightness change (Leuven) and JPEG compression (Ubc). Image from [26].

2.6.5 *A COMPARISON OF SIFT, PCA-SIFT AND SURF PERFORMANCE EVALUATIONS*

The authors of [34] and [35] compare and evaluate various feature detection methods and their descriptors. In [34], Juan and Gwun compare SIFT, PCA-SIFT and SURF. In [35], Mikolajczyk and Schmid evaluate the performance of numerous descriptors, however here we consider only PCA-SIFT and SIFT, as only the latter are relevant in this context. Since both papers follow similar approaches, albeit different, to evaluate the methods, we will explore each paper in turn. One should note that since ASIFT algorithm was proposed only very recently, as yet there exist no evaluations comparing it to other feature extraction algorithms.

2.6.5.1 Evaluation One

Mikolajczyk and Schmid in [35] use the Mikolajczyk database [33] as a dataset for the performance evaluation. The dataset contains six different image transformations, namely rotation, scale change, viewpoint change, image blur, JPEG compression and illumination change. The evaluation criterion proposed is similar to the one used in [24], that is, image pairs with a transformation large enough to introduce some noise. The evaluation criterion is based on the number of correct keypoint matches and the number of false matches for a given image pair; refer to Section 2.6.2 for a description. Following are the experimental results led on the dataset, divided by the image transformation in question.

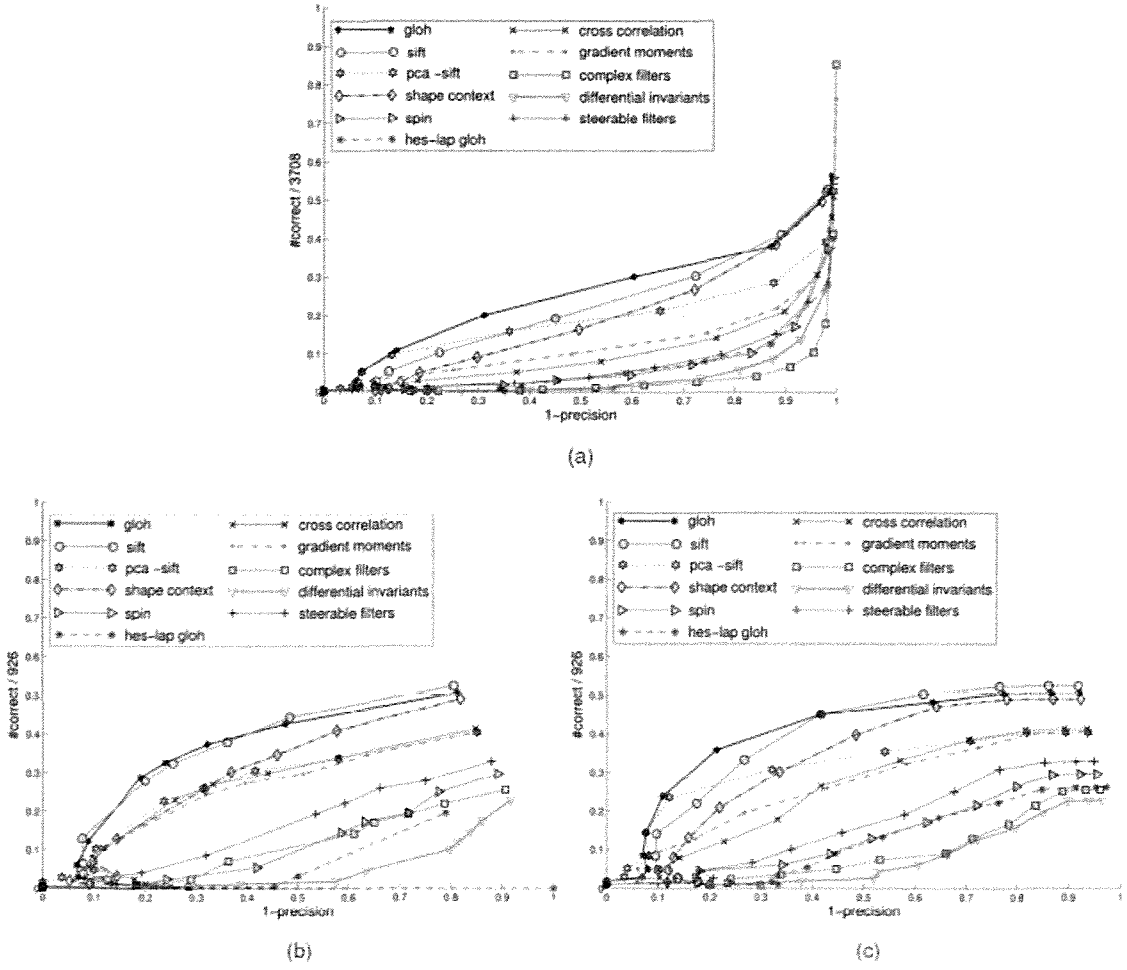


Figure 33: Comparison of different matching strategies. (a) Threshold-based matching. (b) Nearest neighbor matching. (c) Nearest neighbor distance ratio matching. Image from [35].

Schmid and Mikolajczyk first evaluate the following matching approaches:

- Threshold-based matching;
- Nearest-neighbor matching;
- Nearest neighbor distance ratio matching.

The purpose of matching is to obtain only relevant features and ignore irrelevant ones, such as features extracted from the background. The matching of features also enables changes in viewpoint and invariance towards transformations in images. The results of the first test, shown in ***Figure 33***, give similar ranking for all descriptors. As can be seen from ***Figure 33*** (b) and (c) the precision is higher for nearest neighbor-based matching than for the threshold-based approach shown in ***Figure 33*** (a). Nearest neighbor matching selects only the best match below a threshold and rejects all others. This results in less false-positives and a higher precision. This is similar to nearest neighbor distance ratio matching, however the latter penalizes descriptors which have similar matches, i.e. "the distance to the nearest neighbor is comparable to the distances to other descriptors" [35]. Again, this improves the precision. The nearest neighbor-based approaches are best suited for matching, as opposed to searching in a large database. For the latter, the best approach is the similarity criterion, i.e. the distance between descriptors. This approach is used for experiments in this paper.

2.6.5.1.1 Affine Transformations

For affine transformations, performance was evaluated for viewpoint changes of approximately 50 degrees. This viewpoint change "introduces a perspective transformation which can locally be approximated by an affine transformation" [35].

For this section of evaluation, the authors test descriptors performance for different affine region detectors and various scene types. ***Figure 34*** (a) and (b) show the results for a structured scene with Hessian-Affine and Harris-Affine regions while ***Figure 34*** (c) and (d) show the results for a textured scene also using Hessian-Affine and Harris-Affine regions. Textured scenes give better recall than structured ones, and the SIFT-based descriptors perform best in textured scenes. This shows that such scenes require large discriminative power to match. SIFT obtains the best results for the textured images.

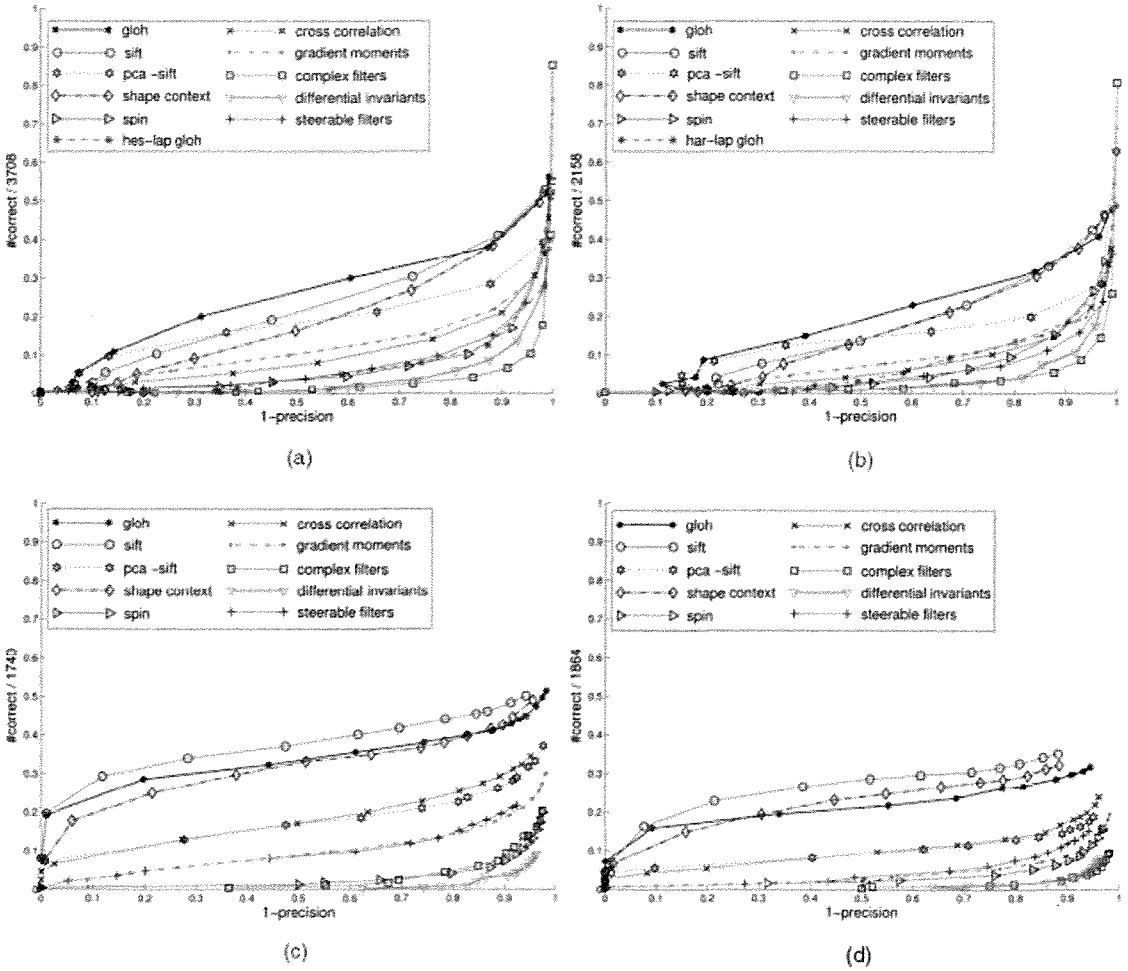


Figure 34: Evaluation for viewpoint changes of 40-60 degrees. (a) Structured scene with Hessian-Affine regions. (b) Structured with Harris-Affine regions. (c) Textured scene with Hessian-Affine regions. (d) Textured scene using Harris-Affine regions. Image from [35].

2.6.5.1.2 Scale Changes

For this stage of evaluation, descriptors are tested using combined image rotation and scale change. Scale changes vary between 2 and 2.5 while image rotation lies within the range of 30 to 45 degrees. ***Figure 35*** (a) and (c) show the performance results for the descriptors computed for Hessian-Laplace regions and the Harris-Laplace regions respectively, detected on a structured scene. ***Figure 35*** (b) and (d) on the other hand use both regions for a textured scene.

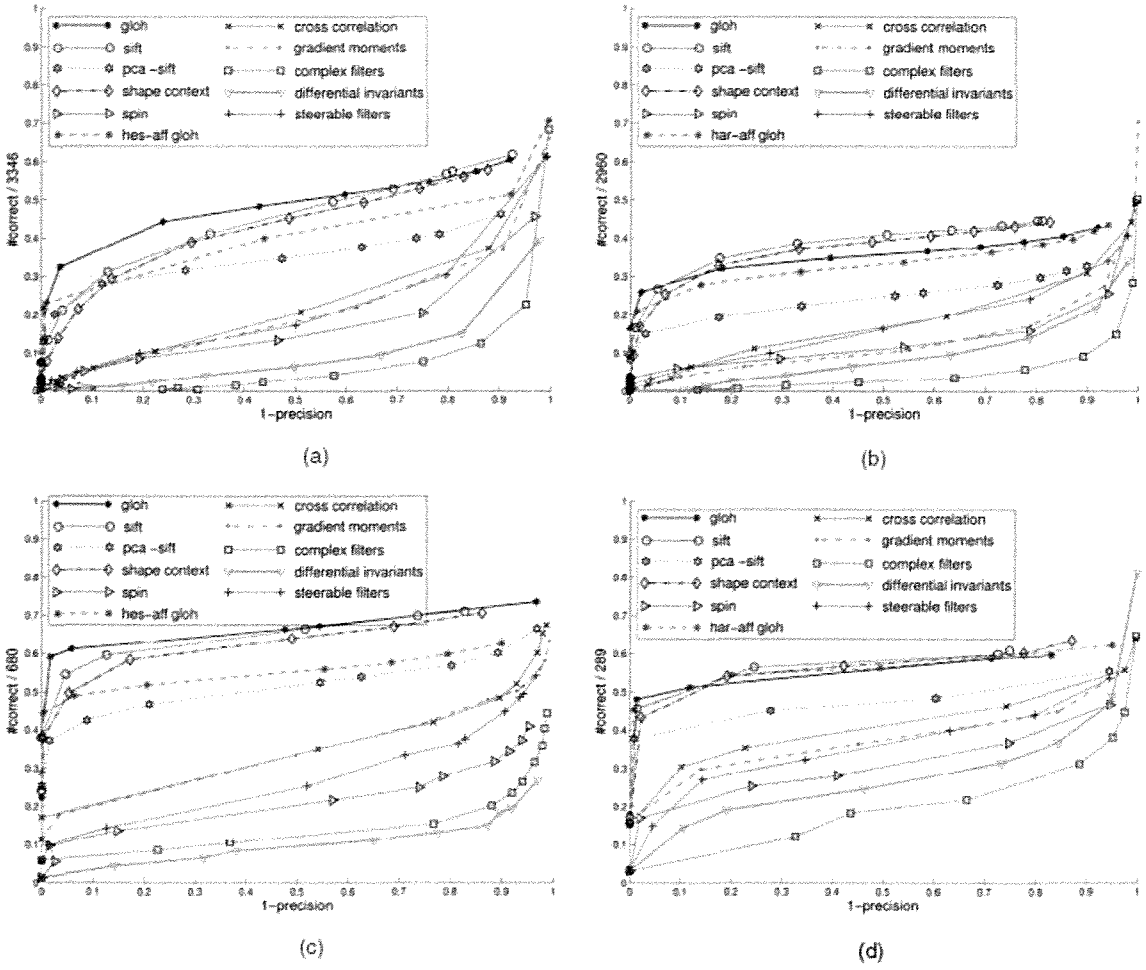


Figure 35: Evaluation for scale changes of a factor 2-2.5 combined with an image rotation of 30-45 degrees. (a) Structured scene with Hessian-Laplace regions. (b) Structured with Harris-Laplace regions. (c) Textured scene with Hessian-Laplace regions. (d) Textured scene using Harris-Laplace regions. Image from [35].

2.6.5.1.3 *Image Rotation*

Images with rotation between 30 to 45 degrees were used for this evaluation. From ***Figure 36*** (a) one can see that all curves are horizontal at similar recall values. This shows similar performance. In ***Figure 36*** (b) SIFT ranks second. The precision is low for all descriptors since they do not capture small variations in texture. This results in many false matches.

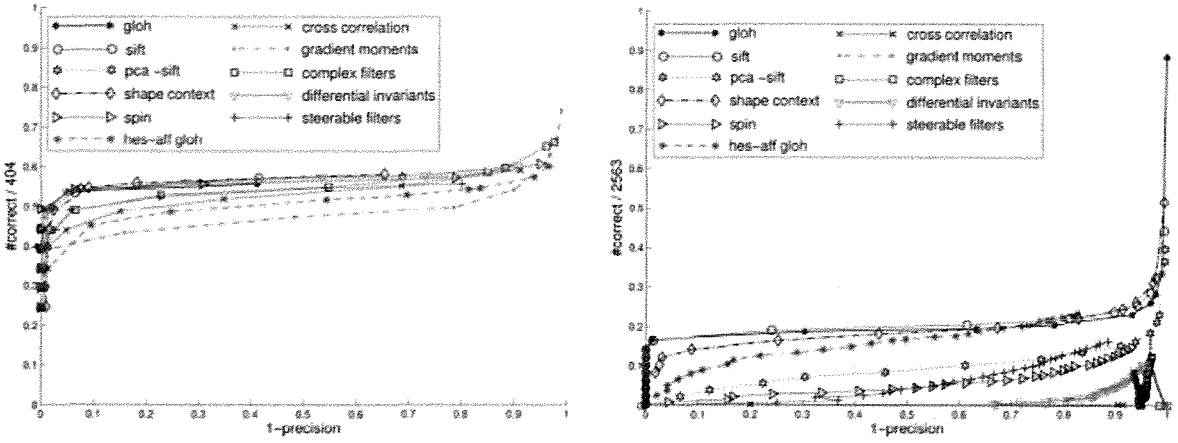


Figure 36: Evaluation for an image rotation of 30 to 45 degrees. (a) Results for structured images. (b) Results for textured images. Image from [35].

2.6.5.1.4 *Image Blur*

This evaluation stage is done using images with a significant amount of blur, introduced by changing the camera focus. ***Figure 37*** (a) shows that PCA-SIFT achieves higher scores than SIFT. The opposite is true in ***Figure 37*** (b). Results show that all descriptors are affected by blur in images, though since it is difficult to model naturally-occurring blur, artificially generated blurred images tend to be overly-optimistic.

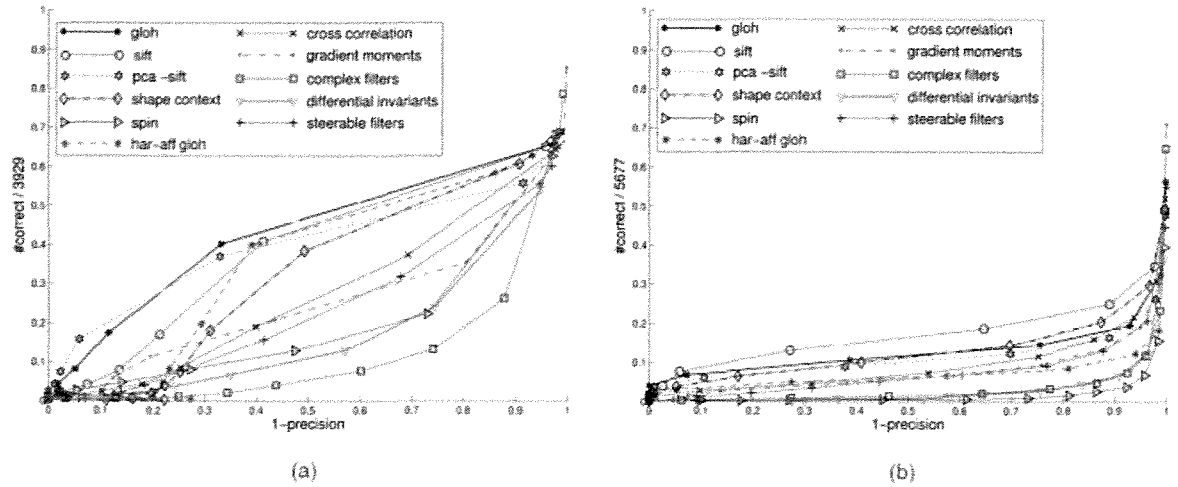


Figure 37: Evaluation for blur. (a) Results for a structured scene. (b) Results for a textured scene. Image from [35].

2.6.5.1.5 *JPEG Compression*

The influence of JPEG compression for a structured scene is evaluated in this stage. The quality of the transformed image used is 5% of the original one. As can be seen from FIGURE 38, descriptor performance increases when precision is decreased, for all descriptors. PCA-SIFT obtains the best score for a low false-positive rate while SIFT obtains the best score for a false-positive rate above 0.2.

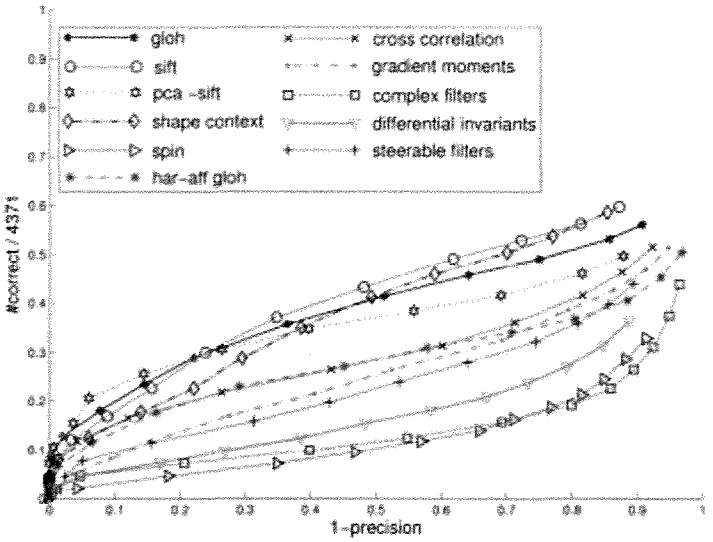


Figure 38: Evaluation for JPEG compression. Image from [35]

2.6.5.1.6 *Illumination Changes*

Images for this evaluation were obtained by changing camera settings. The results are shown in ***Figure 39***. SIFT obtains higher scores than PCA-SIFT.

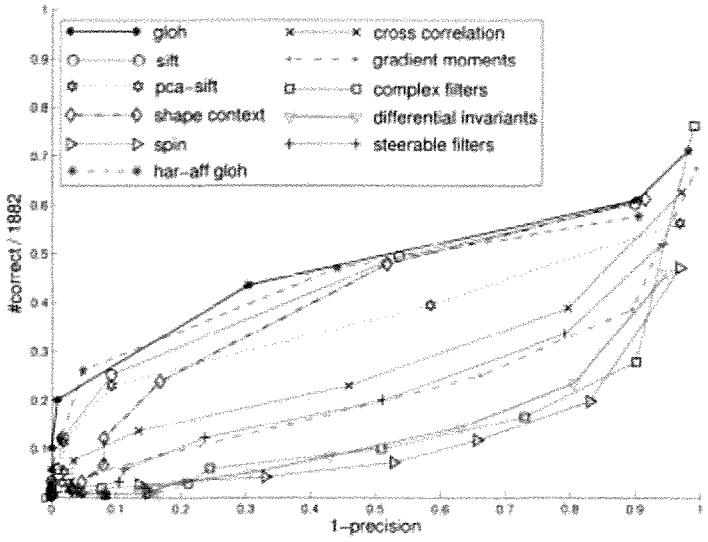


Figure 39: Evaluation for illumination changes. Image from [35].

Mikolajczyk and Schmid also perform a matching example for images with a viewpoint change of more than 50 degrees. For a fixed number of 400 nearest neighbor matches, SIFT obtains a higher score than PCA-SIFT, obtaining 177 and 139 nearest neighbor correct matches respectively. Recall and precision values for SIFT are 0.24 and 0.56 while for PCA-SIFT the values are 0.19 and 0.65.

2.6.5.2 Evaluation Two

Juan and Gwun in [34] compare SIFT, PCA-SIFT and SURF. They use K-nearest neighbor (KNN) and Random Sample Consensus (RANSAC) with the aim of evaluating the methods' application in recognition. KNN is used to find matching keypoints while RANSAC is used to reject inconsistent matches. Similar to [35], the feature detectors are compared for scale changes, rotation, image blurring, change in illumination and affine transformations. Methods' performance is ranked according to repeatability measurement and the number of correct matches.

The repeatability measurement is defined as "a ratio between the number of point-to-point correspondences that can be established for detected points and the mean number of points detected in two images:

$$r_{1,2} = \frac{C(I_1, I_2)}{mean(m_1, m_2)}$$

Equation 4: Repeatability measurement equation. Equation from [34].

Where $C(I_1, I_2)$ denotes the number of corresponding couples, m_1 and m_2 means the numbers of the detector. This measurement represents the performance of finding matches" [34].

RANSAC is used as an evaluation measurement to reject inconsistent matches. An inlier can be considered a point which has a correct match in the input image. An outlier, on the other hand, can be considered the equivalent of a false-positive. Thus, the goal is to obtain the inliers and reject outliers. "The probability that the algorithm never selects a set of m points which are all inliers is 1-p:

$$1 - p = (1 - w^m)^k$$

Equation 5: Equation of the probability that the RANSAC algorithm never selects a set of m points which are all inliers. Image from [34].

Where m is the least number of points needed for estimating a model, k is the number of samples required and w is the probability that the RANSAC algorithm selects inliers from the input data" [34].

2.6.5.2.1 Processing Time

The evaluation of time is relative, depending on image size, quality, type (e.g. scenery or texture) and algorithm parameters. Thus, such an evaluation only shows the time the three algorithms take relative to each other. The dataset used was the Graffiti dataset, with 300x240px image

sizes. The algorithms' parameters were set as defined in the original papers [26] [24] [18]. Time was counted for the complete processing, including feature detecting and matching. ***Figure 40*** shows the results; SURF is the fastest while SIFT is the slowest, yet finding the most matches.

Items	SIFT	PCA-SIFT	SURF
total matches	271	18	186
total time (ms)	2.15378e+007	2.13969e+007	3362.86
10 matches' time(ms)	2.14806e+007	2.09696e+007	3304.97

Figure 40: Processing time comparison. Image from [34].

2.6.5.2.2 Scale Changes

This experiment evaluates the algorithms' scale invariance. As can be seen from ***Figure 41*** and ***Figure 42***, SIFT and SURF perform much better than PCA-SIFT when the scale change gets larger.

Figure 41: Scale changes comparison. The first result is SIFT (10/66 matches), second is SURF (10/26 matches). The last two images show the results of PCA-SIFT in scale changes. Image from [34].

Data	SIFT	PCA-SIFT	SURF
1-2	41	1	10
3-4	35	0	36
5-6	495	19	298
7-8	303	85	418

Figure 42: Scale changes comparison showing total number of matches for each method. Image from [34].

2.6.5.2.3 Image Rotation

The next experiment is aimed to show the effect of rotation on the three algorithms. SIFT is the method which detects the most matches and is the most stable to rotation. SURF, on the other hand, does not work as well, finding the least number of matches and achieving the least repeatability. Though PCA-SIFT found only one correct match of the first ten matches, it can be improved, thus it is better than SURF. ***Figure 43*** and ***Figure 44*** show the results.

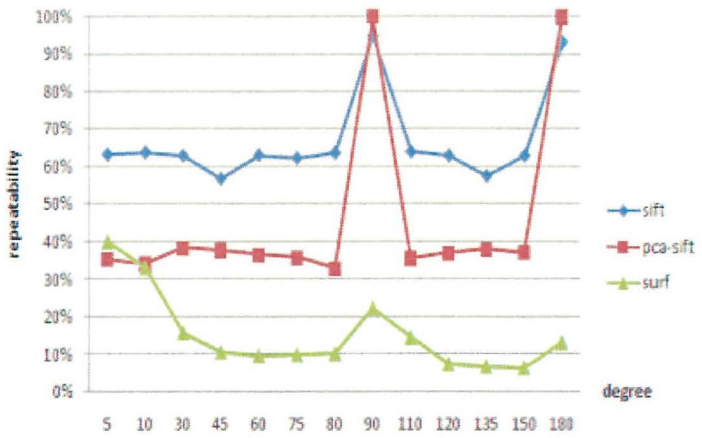


Figure 43: Rotation comparison showing the repeatability of rotation. Image from [34].

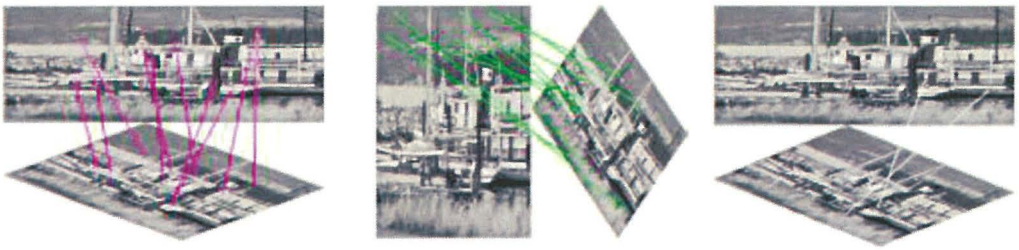


Figure 44: Rotation comparison with rotation degree of 45. The first two images show the first ten matches of SIFT (10 correct /10) and PCA-SIFT (1 correct /10), the result of SURF (2 correct /2) is the total number of matches. Image from [34].

2.6.5.2.4 Image Blur

This experiment uses a blur similar to a Gaussian. The radius of the blur ranges from 0.5 to 9.0. As shown in ***Figure 45***, SURF and PCA-SIFT perform well, however as the radius of the blur gets larger, SIFT shows the best performance, while SURF finds a small number of matches and PCA-SIFT finds a larger number of matches, but fewer correct ones. This can be seen in ***Figure 46***.

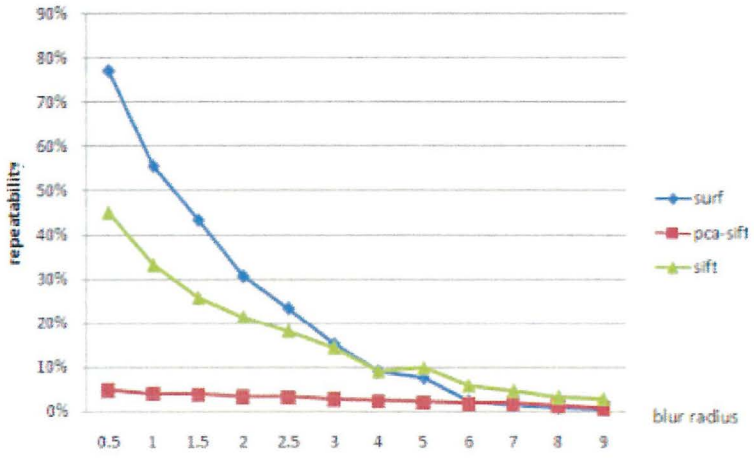


Figure 45: Blur comparison showing the repeatability of blur. Image from [34].

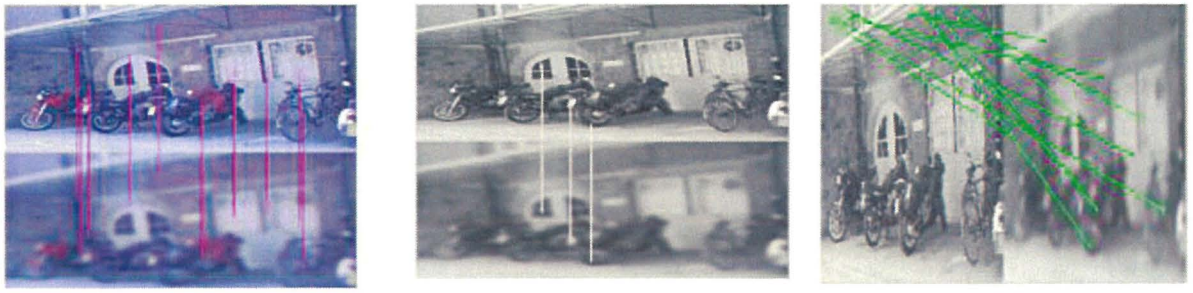


Figure 46: Blur comparison. From left to right: SIFT (10 correct /10), SURF (3 correct /3), PCA-SIFT (1 correct /10), with blur radius of 9. Image from [34].

2.6.5.2.5 Illumination changes

The fifth experiment evaluates the effect of illumination change on the methods. As shown in ***Figure 47***, SURF has the highest repeatability at 31%.

Data	SIFT	PCA-SIFT	SURF
1-2	43%	39%	70%
1-3	32%	34%	49%
1-4	18%	27%	25%
1-5	8%	18%	6%
1-6	2%	9%	5%
average	21%	25%	31%

Figure 47: Illumination changes comparison showing repeatability and the average of repeatability. Image from [34].

2.6.5.2.6 Affine transformations

The last experiment evaluates the "methods' stability of affine transformations" [34]. The change of viewpoint in the images used is approximately 50 degrees. As can be seen in ***Figure 48***, SURF and SIFT have a good repeatability with a small viewpoint change, however with a larger viewpoint change, SURF detects 0 matches and PCA-SIFT performs better.

Data	SIFT	PCA-SIFT	SURF
1-2	47%	15%	54%
1-3	37%	12%	33%
1-4	22%	12%	12%
1-5	7%	10%	2%
1-6	0%	9%	0%

Figure 48: Affine transformation comparison showing the repeatability. Image from [34].

Method	Time	Scale	Rotation	Blur	Illumination	Affine
SIFT	common	best	best	best	common	good
PCA-SIFT	good	common	good	common	good	good
SURF	best	good	common	good	best	good

Figure 49: Conclusion of all experiments. Image from [34].

Figure 49 shows the results of all led experiments. It indicates that there is no best method for all image transforms. Obviously, the experiments led are not constant, as a change in an algorithm or the matching method used can change the results. ***Figure 49*** shows that PCA-SIFT would require an improvement to handle blur and scale deformations. SURF is fast and performs quite well in most situations, however its performance degrades with large rotations. SIFT is stable in all experiments except for time and changes in illumination.

2.7 Similar Systems

2.7.1 Mobile Visual Aid Tools for Users with Visual Impairments

The authors of [36] explain 'MobileEye'; a mobile based software suite intended for the use of visually impaired persons in their daily activities. MobileEye consists of four functions, namely:

- "A color channel mapper which can tell the visually impaired different colors;
- A software based magnifier which provides image magnification as well as enhancement;
- A pattern recognizer which can read currencies;
- A document retriever which allows access to printed materials." [36]

Since it is the most relevant for the subject of this thesis, here we focus on describing the pattern recognizer. The aim of the latter is to recognize currency bills. This is done by processing the image in real-time, since a visually impaired user could find it hard to take a focused picture of a currency note.

First, a very fast classifier is used to filter the images and locate areas of the images which probably do not contain the target feature area (the part of the bill showing the number denomination). This step is inspired by the Viola-Jones face detector [37] and SURF [26], which both implement box filters for the purpose of object and feature detection. This pre-classification stage was found to speed the algorithm by 20 times on a camera phone [36]. Then, 32 local pixel pairs are used to form weak classifiers. Such pairs have a relatively stable relationship in that one pixel is usually brighter than the other. Ada-boost is then used to train strong but fast classifiers from the weak classifiers. Local pixel pairs provide the advantages of being robust to noise, blur and change in illumination.

The software suite was designed with the end-users of MobileEye in mind. Each operation is guided by vocal instructions and other details include the automatic exit of the software if it is idle for more than two minutes, to save battery power. A user evaluation was also carried out where the results presented showed that visually impaired users managed to recognize a bill in

21.3 seconds on average. No false positive was raised by the currency reader during the experiment.

2.7.2 *Mobile Museum Guide based on fast SIFT recognition*

[38] explores the feasibility of a mobile based pattern recognition system serving the purpose of a museum guide system; a camera-enabled phone is used to recognize paintings in art galleries. Simply, a user takes a picture of a particular painting and image processing technology is used to recognize the painting. The user is then provided with information regarding the painting in question, such as artist, title, and historical context.

The architecture of the explored system is based on the client-server approach, where the client only acts as a peripheral device used to take and send data (picture) and receive the results. This type of architecture was used for several reasons. First of all, the CPU of mobile phones has low computational power, thus running feature extraction on the phone would result in unbearably long times for the user to receive a result. Besides, on a server, even low resolution images would return good results.

After thoroughly evaluating the SIFT and SURF algorithms, SIFT was found to be most suitable for the purpose of this system. Both SIFT and SURF algorithms' best benefit for this purpose is the use of local features, since global features would require the extraction of objects of interest from the background. The authors explored an approximated SIFT; a k-means based clustering approach coined SIFT fast. This was implemented with k=5. Evaluation showed that some performance deterioration was obvious, however since the implementation in question mainly concerns frontal views, SIFT fast gave an acceptable trade-off between speed and performance.

2.7.3 *Scene recognition with camera phones for tourist information access*

The authors of [39] present Snap2Tell, a prototype system which provides a scene description based on an image taken of the scene. For this system they use the STOIC 101 database and implement an innovative approach towards pattern discovery. The STOIC 101 database is made up of 101 Singapore tourist points of interest with a total of 5278 images. The images were taken from various distances and perspectives with occlusion and clutter. GPS coordinates were also recorded.

The authors propose "a new pattern discovery algorithm to learn image patches that are recurrent within a scene class and discriminative across others" [39]. This approach generates positive training patches for discriminative learning. Support Vector Machines (SVMs) are used as discriminative classifiers. Multi-scale uniform sampling is used to extract patches from images instead of interest point scheme as the former has an advantage over random and uniform samplings when the sampling is dense enough.

A thorough experimental evaluation was led using patch features such as color histograms in RGB, HSV or HS color channels, linear edge histograms and combined color and edge histograms. This was done since the authors believed that color and edge features were the most relevant for the STOIC database. Feature vectors were compared using simple city block distance metric, while color and edge features were combined linearly using equal weights in the SVM. Using GPS coordinates for location priming proved to considerably increase the recognition rate.

2.7.4 *Geo-contextual priors for attentive Urban object recognition*

In [40] the authors propose the exploitation of "geo-information in association with visual features to restrict the search within a local context". They describe the embedding of this association in a general system implementation of an Attentive Machine Interface (AMI). The AMI enables the contextual processing of information gathered from various sensors in a probabilistic framework.

For the implementation, the authors chose an Informative Features Approach which applies local density estimations. By determining the posterior entropy, computed from a normalized histogram of object votes, local information content is made explicit with regard to object discrimination. Thus images which do not contain objects of interest are discarded. This proposed recognition process is "characterized by an entropy-driven selection of image regions for classification, and a voting operation" [40].

For urban object recognition, the object hypothesis is determined from the appearance of the image and the GPS-based positioning; first, the user position is defined within a given radius, then distances between the user location and points of interest is estimated. These distances are then weighted. The recognition and the geo-services processes are computed in parallel, and the results are integrated using Bayesian decision fusion.

The use of geo-contextual information contributed to the improvement in performance of the system, as it enabled a "meaningful selection of expected object hypotheses" [40].

2.8 Conclusion

In this chapter we explored and described various methods and concepts which, put in the context of this thesis, help the user understand better its contribution. The use of the GPS system was explained, as well as digital image representation and image transformations. Various object recognition approaches were explored, and the SIFT, SURF, PCA-SIFT and ASIFT algorithms were explained and evaluated. A number of similar systems were also explained to aid the user understand the current object recognition systems.

Chapter 3

DESIGN

The purpose of White Cane Device is to provide a travelling aid to visually impaired persons. The system should enable the latter to take an image using a camera and GPS equipped mobile phone. The query image should be sent to a server (through a webservice) which extracts features and GPS tags from the image and implements object recognition against a database of features. If an object is recognized, the result should be returned back to the user. An abstract overview of the system is shown in ***Figure 50***. In this chapter we describe the proposed design for the implementation of White Cane Device. This includes the various modules and their purpose, as well as the flow of data. We also propose criteria for evaluating the system.

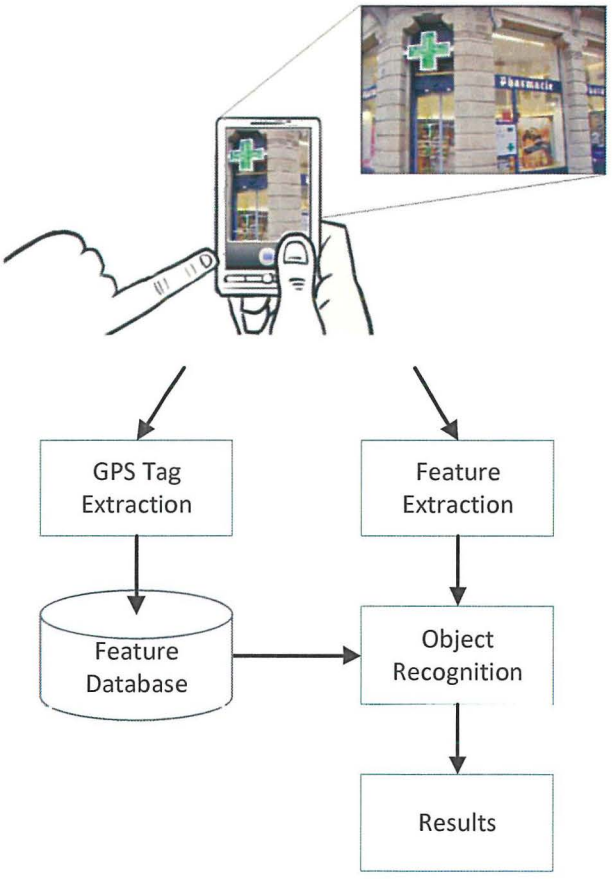


Figure 50: Overview of system

Two approaches can be chosen for the system architecture, namely a client approach and a client-server approach. A client-server architecture is the best approach for this system for

several reasons. The main reason is that feature extraction algorithms require large computational power. The execution of such an algorithm on a mobile device would take much longer than on a server, due to the limited processing power of mobile devices when compared to a server [3]. Also, the feature database would require a large amount of space, which is not so feasible on a mobile device. A disadvantage of using the client-server architecture is the requirement of a telecommunication means such as WIFI or 3G.

Using a client-server architecture means the client can act as a periphery device, used only to acquire the image and send it over to the server. This would also encourage portability, as it would be much easier to create various mobile clients for different mobile operating systems. One should note that White Cane Device does not make use of additional hardware installed in location such as Bluetooth senders. This improves the scalability of the system. The systems developed in [39], [40] and [38] all use the client-server approach for mobile based object recognition. The latter systems are explored in Section 2.7.

3.1 System Design Overview

The required system will be divided into three main components; the client, the server and the database. These components can be seen in ***Figure 51***, which shows a level 0 DFD of the system. The components' functions are as follows:

3.1.1 Client Component

The client component is a mobile-based application which allows a user to take a picture of his/her surroundings and then send it over to the server component. An activity diagram of the client component is shown in ***Figure 52***. When the user takes an image, the geo-tagging option should be enabled. Geo-tagged images would enable the optimization of the recognition process. As shown in ***Figure 20*** in Section 2.6.1, the reliability of the matching process decreases as the number of features in the database increases. The optimization process is further explained in Section 3.1.3. As shown in [40], the use of location priming (GPS coordinates) can substantially increase the recognition rate of an object recognition system. By using GPS coordinates, the search space can be reduced to features extracted from images taken within a specified radius from where the query image was taken.

Keeping in mind the end users of the client component, the user interface of the mobile application is designed to be very simplistic, with as few operations as possible to capture an image. Also, the application implements voice prompts to guide the user as to what is expected from him/her.

[3] Consider the example of an HP Pavilion dv5 laptop and an HTC Desire mobile phone. The former has an AMD Athlon X2 Dual-Core QL-60 1.90 GHz CPU with 3.00 GB RAM while the latter has a Qualcomm Snapdragon 1 GHz CPU and 1GB RAM.

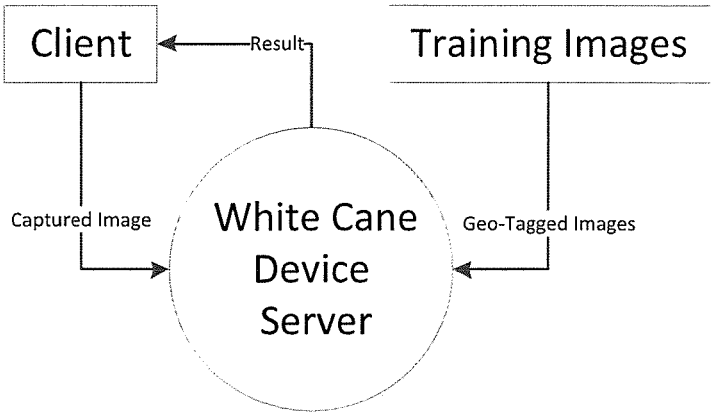


Figure 51: DFD Level 0

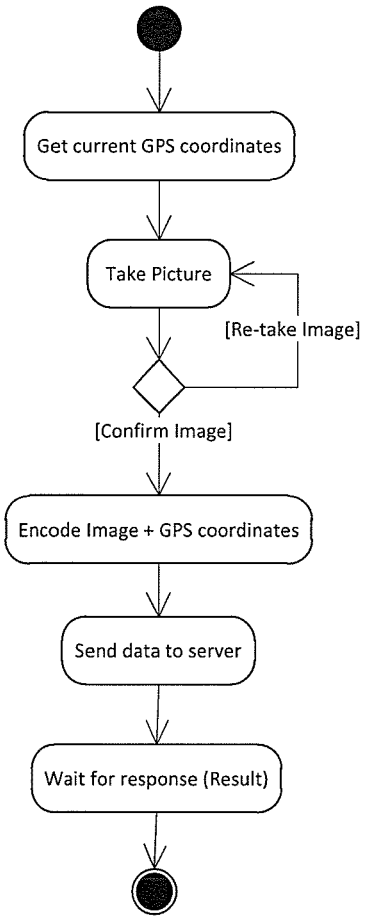


Figure 52: Activity Diagram for Client component

3.1.2 *Server Component*

The server component is made up of two processes; the training process and the query process. The level 1 DFD shown in ***Figure 53*** shows data required from these processes, and also any results returned. Before a query is performed, the training process should be run at least once to populate the database, but it could be executed for as many times as desired.

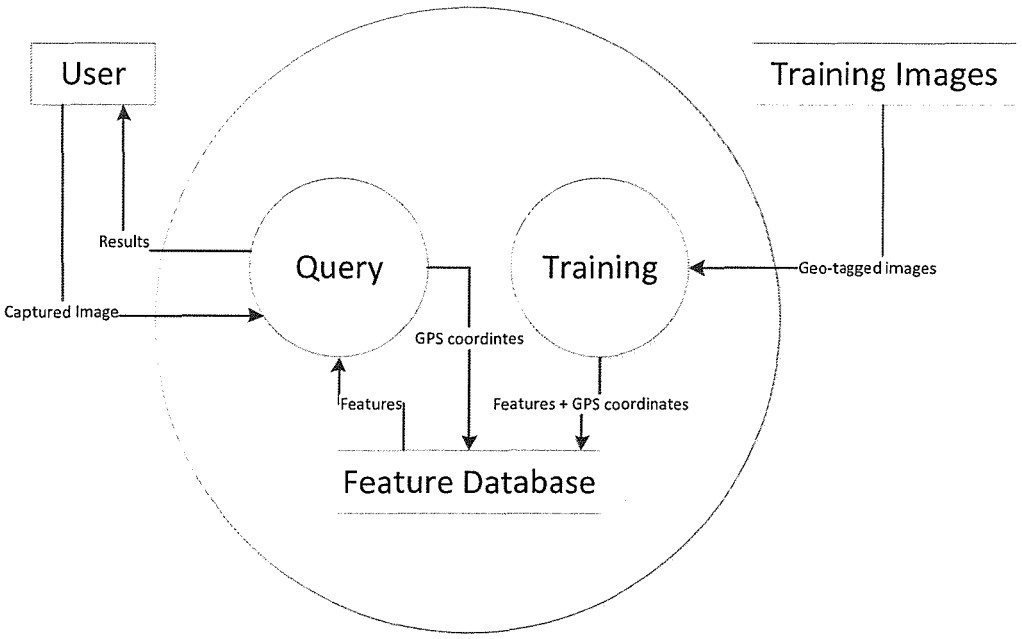


Figure 53: DFD Level 1

3.1.2.1 Training

The training process enables a user to select a number of images from the local machine. This is done using a very simple user interface consisting of a window with a button, which opens up a dialog box. Though this process is not directly aimed for the use by visually impaired persons, the use of a screen reader should enable such persons to use it easily. The purpose of the training process is to populate the feature database with features, which are then used by the query process for object recognition. ***Figure 54*** shows the level 2 DFD of the training process. For each image, the training process extracts the features using a feature extraction algorithm, and the GPS tags. Each image results in a large number of features, and generally the larger the image, the more features are extracted. The number of features however also depends on the subject of the image. The feature descriptor is then serialized to the feature database, along with the GPS coordinates and the name of the image from which the feature was extracted.

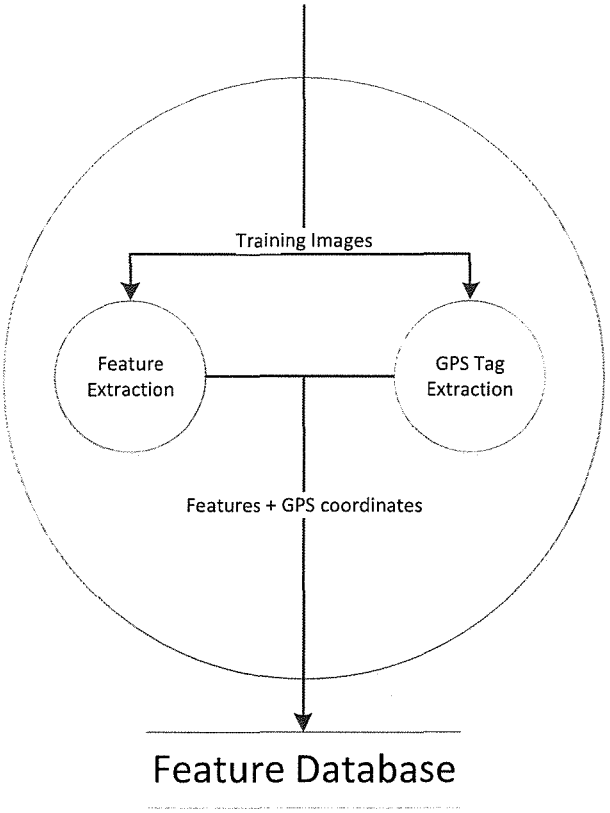


Figure 54: DFD Level 2 of Training Process

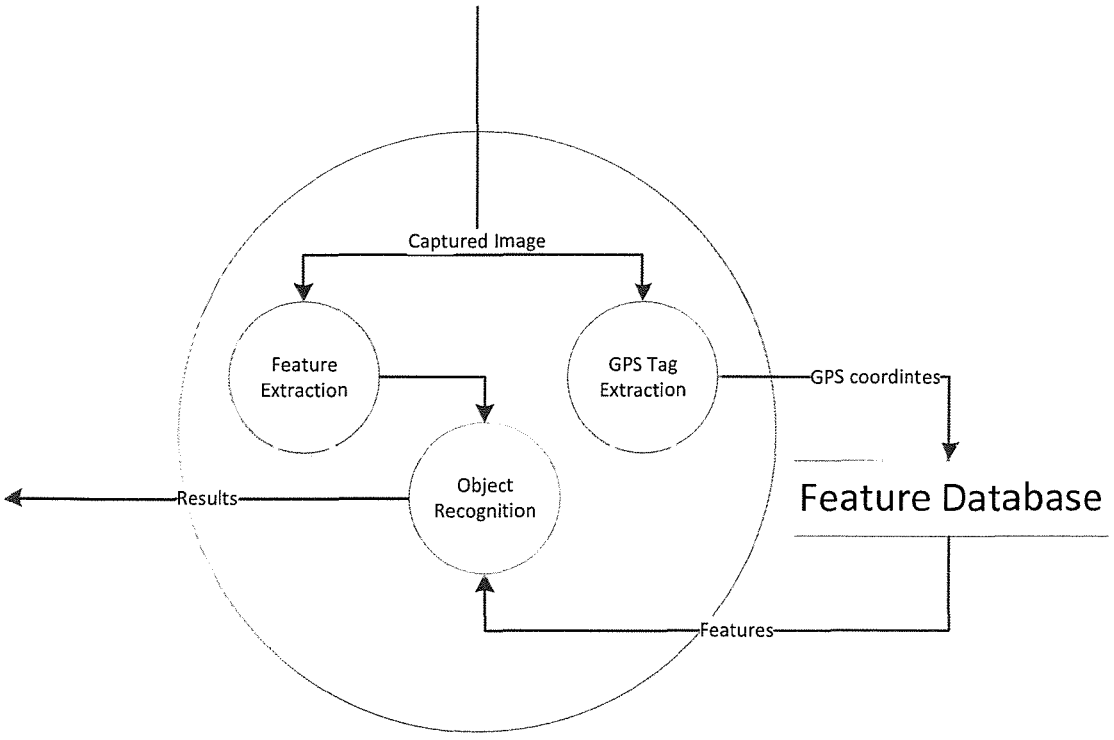


Figure 55: DFD Level 2 of Query Process

3.1.2.1.1 Feature Extraction

Since the scope of this thesis doesn't allow for controlled lighting, scaling and rotation, the ideal approach for the feature extractor is feature-based. Feature-based algorithms are the most promising to achieve the desired objectives since they are the most robust regarding scale, lighting and changes in perspective, and their greatest benefit is the use of local features. The local feature approach, in contrast with the global feature approach, selects only distinctive features from the image in question. The global feature approach considers the image as one entity. In [38] both SIFT and SURF algorithms were evaluated in a fully implemented prototype mobile-based system to find the best performing algorithm. The SIFT algorithm was also implemented in [40].

3.1.2.2 Query

The query process is the crux of the system. This process receives a geo-tagged image, as depicted in ***Figure 55***. The GPS coordinates are extracted from the image, and the feature extraction algorithm is implemented as described in Section 3.1.2.1.1. The GPS coordinates are then used to formulate a query to retrieve particular features from the feature database. The latter features should be from images taken within a particular radius from where the query image was taken. ***Figure 56*** shows an activity diagram describing the query process.

3.1.2.2.1 Object Recognition

This process accepts the features extracted from the query image, along with the features loaded from the feature database according to the GPS coordinates of the query image. These features are used by the matching algorithm to find if the database contains an image depicting the same object as the query image. The name of the matching image, if any, is then returned to the client component.

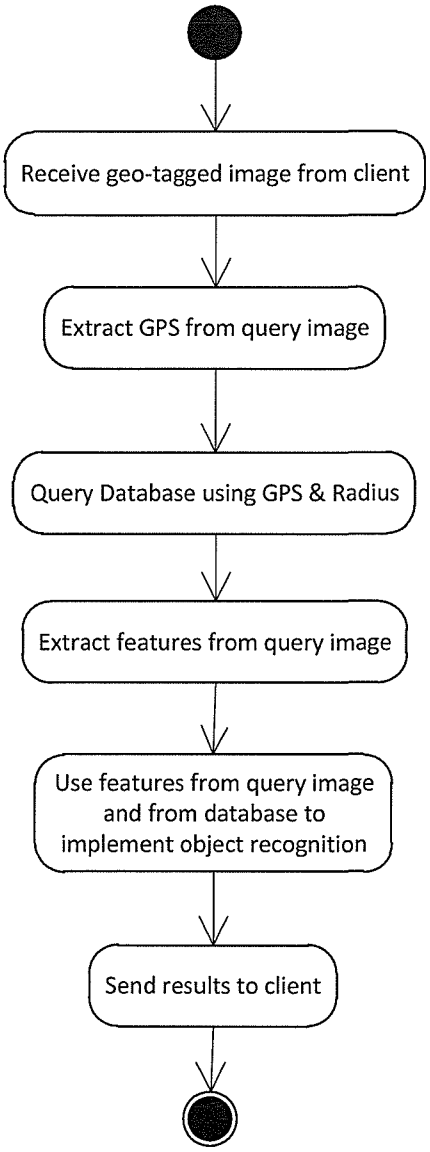


Figure 56: Activity diagram depicting the Query process

3.1.3 *Database Component*

The database is made up of a single table containing features extracted from the training images, and other relevant data. This includes:

Name	Description
Id	The primary key of the table, used to uniquely identify each feature in the database.
Image Name	The name of the image the feature was extracted from. A single image will have a large number of features.
Descriptor	The descriptor of the feature in question. This is used for the matching process.
GPS	The coordinates of the image that the feature was extracted from.

Both the query and the training processes interact with the database, as shown in ***Figure 53***. The training process passes both the GPS coordinates and the features extracted from the training images, which are appended to the database. The query process, on the other hand, passes the GPS coordinates of the query image as a parameter to create a query. This query enables the loading of features which were extracted from images taken within a particular radius from where the query image was taken. These features are then returned to the query process.

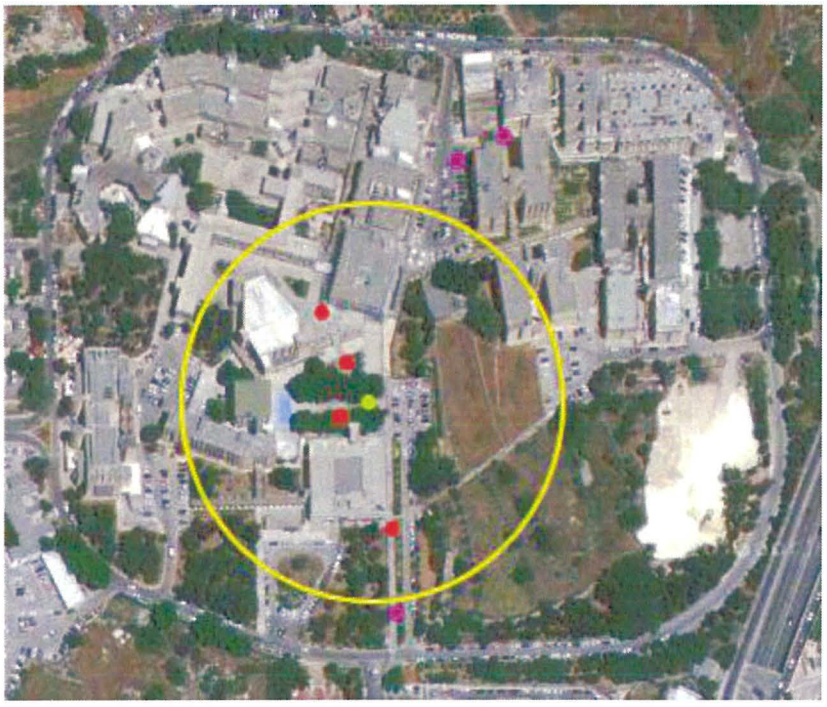


Figure 577: Training images within a given radius

Figure 577 shows an aerial view of the University of Malta. Assume the yellow point is the location where the query image was taken, and that all other points are the locations of training images. If the radius value (the yellow circle) is set, for example, to 20m, then all images taken in

locations which fall within the specified radius are loaded from the database. Thus, the red points shown in the image are loaded, while the pink ones are not.

3.2 Evaluation

For the evaluation of an object recognition system, as in the case of White Cane Device, one must mainly evaluate the correctness, accuracy and response of the latter. The assessment of a system is divided into four parts [41]:

1. **Accuracy Requirements:** Here a system is evaluated on its capability of giving accurate results and also the ability to anticipate the degree of certainty of the accuracy of its results. In [38] Cumulative Match Characteristic (CMC) curves were used to this purpose.
2. **Samples Constraints:** The training set used must be carefully considered before assessing the system's performance. Image aspects such as perspective and resolution have an impact on the results.
3. **Speed:** Since for the purpose of this thesis, the results are required in real time, computational speed plays a vital part for the evaluation of this system.
4. **User Interface:** The system's user interface should be evaluated to assess the software usability.

3.3 Conclusion

In this chapter we described various aspects of the design required for the implementation of White Cane Device. The best architecture for this system is the client-server architecture. The various components and modules which make up the system were described, as well as data passed between modules. We also proposed evaluation criteria for the final evaluation of the system.

Chapter 4

IMPLEMENTATION

In this chapter we will define the methods and approaches used to reach the specified objectives of this thesis, according to the design specified in the previous chapter. This chapter includes detailed descriptions of how the various modules and components of White Cane Device were implemented.

4.1 CLIENT COMPONENT

The client component, which is the component used by the visually impaired user, was implemented on an HTC Desire mobile phone, a GPS and camera enabled touchscreen smartphone. The Desire has a 1 Ghz Qualcomm Snapdragon processor, a 5 megapixel camera with auto focus and flash and the Android version 2.2 operating system. The client component is a mobile application which was developed using the Android version 2.2 API level 8. This means that the mobile application is compatible with all phones running the Android operating system version 2.2 or later. Monodriod, a software development kit allowing developers to use C#.NET to create mobile applications for Android-based devices, was also considered due to more experience with C#.NET. However, since Monodriod is fairly recent, there is little support for developers, and scarce example code.

Figure 58: HTC Desire. Image from [50]

4.1.1 *User Interface*

The most important thing we kept in mind for the implementation of the user interface is the end user, that is, the visually challenged person. Such a person can only navigate a touchscreen phone by using the TalkBack function, a screen reader which speaks out as the user navigates. The screen reader reads out menus, application names, and any actions taken by the user such as a click. The TalkBack function provides spoken feedback to the user if the Accessibility function is enabled. This can be done through the settings menu.

As can be seen in ***Figure 58***, the Desire has 3 buttons and an optical track ball. The buttons, from left to right, have the following function:

- Home screen;
- Menu;
- Left side of button: Back, Right side of button: Search.

The optical track ball can be used to navigate through the phone without using the touch screen; a press of the trackball is equivalent to a click on the touchscreen. Buttons provide easy interaction with the phone, as the user can easily feel and distinguish the buttons. With the latter things in mind, the user interface was implemented as a simple application which when run starts up the camera and requires only two key presses from the user to get a result. The TalkBack function was also implemented in the application to provide prompts and guidance to the user, whether the Accessibility function of the phone is enabled or not.

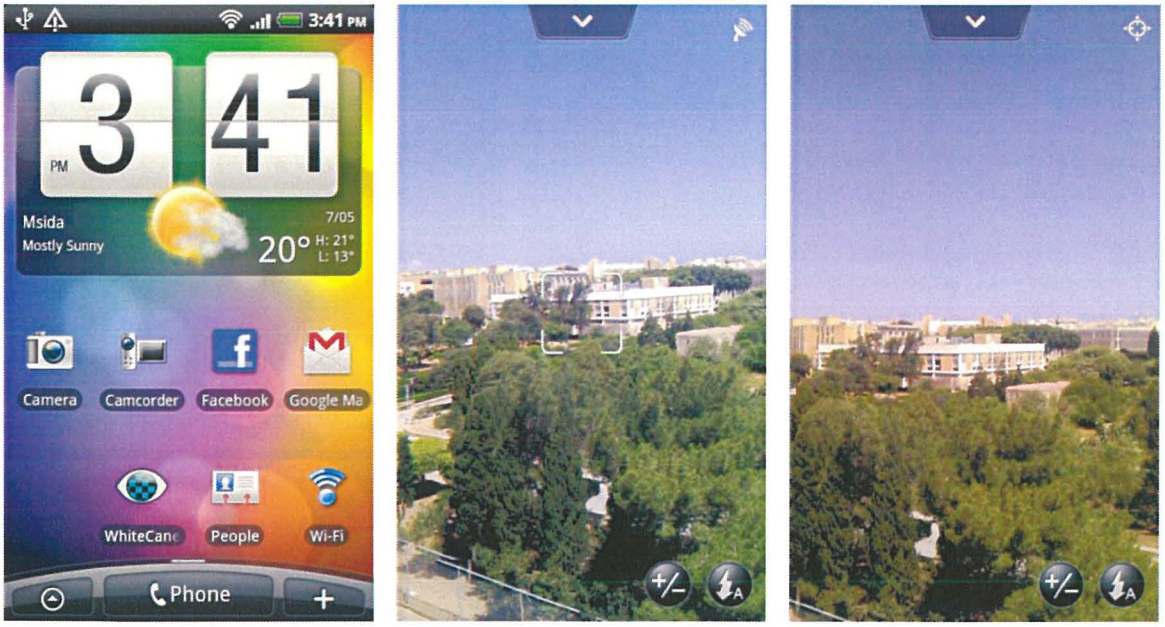


Figure 59: Screenshots of Application. From Left to right: Home screen, Running application - searching for GPS, Running application - GPS locked

The middle and the rightmost screenshots in ***Figure 59*** depict the running application. The icon on the top right corner shows that the geo-tagging setting is enabled. This can be done through

the camera settings. When the icon changes to the one shown in the rightmost screenshot, the phone has obtained the current GPS coordinates.

On start-up, the application starts the camera and prompts the user to take a picture. At this point, if the user wants to exit he/she only requires pressing the back button twice; a prompt will inform the user that the operation was cancelled. A user can take a picture in two ways; either by pressing and holding anywhere on the touchscreen (the holding enables the camera to auto-focus), or by pressing the optical trackball. If on start-up the WIFI is disconnected, the application prompts the user to wait until a connection is established. When an image is taken, the user can decide to send the image to be recognized, or else take another image. The option to take another image is given in case the user feels he moved the mobile phone while taking the picture, as this would result in a blurry image. The application provides two buttons, shown in ***Figure 60***, for the said decision. In the case of this thesis, however, these buttons are not user friendly. The user can instead use the optical trackball to navigate between both buttons. By 'swiping' the trackball to the right or left, the focus passes from one button to another, and a press of the optical trackball confirms the image or restarts the camera to take another picture. The focus is on the 'done' button by default. When the user confirms the image, the user is prompted to wait for a response. After hearing the response, the user can exit the application by pressing the back button. One should note that the image to be sent to the server should be of a small size. In this implementation we use 384x640px images. The picture resolution only needs to be set once, then it remains as set for any subsequent runs of the mobile application.

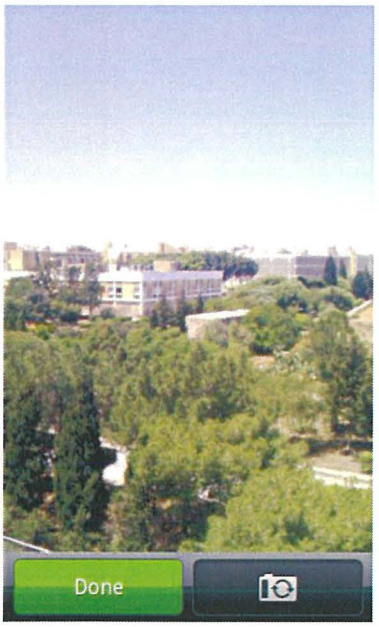


Figure 60: Screenshot on Image taken

4.1.2 BACKEND

The backend of the application is made up of a single class, CameraActivity, which extends the Activity class and implements OnInitListener. CameraActivity has two main functions, namely

taking a picture and sending it over to the web service. Some of the code used in the backend was adapted from a code sample[4]. The following are three main methods implemented to achieve the required results:

- **onCreate**
  This method is called on start-up of the application to initialize the latter, and overrides the Activity `onCreate` method. The `onCreate` method first sets the layout using the `setContentView` method. This method is used to set a layout resource, defined as an xml file, to define the user interface. In this case, the layout is set to take the whole screen of the phone. The method then checks the current state of the WIFI, and turns it on if it was not enabled. Thus, the user does not have to turn it on manually.

  Within the `onCreate` method, a `TextToSpeech` object is then initialized. This makes a call to the `onInit` method, described later on in this section. The last part of this method is used to create a new camera intent. An intent is a type of operation to be performed. The default camera intent only returns a small bitmap image of the picture taken. Thus, the `putExtra` method is called, which saves the full sized image to a given URI with a given file name. To be unique for each image captured, the images are given the current system time in milliseconds using the *currentTimeMillis* method. Finally, the `startActivityForResult` method is called. The latter method calls the `onActivityResult` method once the camera intent activity is complete.

```
Intent cameraIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
cameraIntent.putExtra(MediaStore.EXTRA_OUTPUT, mCapturedImageURI );
startActivityForResult(cameraIntent, PICTURE_ACTIVITY);
```

- **onActivityResult**
  This method is called upon completion of the camera intent described above. First, an if-statement is used to check if the result of the activity was a 'cancelled operation' or a 'confirmed image'. If the result is a confirmed image, the URI of the image just saved is obtained. Then, the GPS coordinates are extracted from the image EXIF tags, however if no GPS coordinates are available, null values are set. The coordinates are in string form for each of the latitude and longitude (DDD/1, MM/1, SSS/100), latitude ref (N or S) and longitude ref (E or W). The image is then compressed into PNG format and converted into a byte array. It is important to note that the GPS tags are extracted from the image before the compression because PNG images do not support such tags. `PropertyInfo` objects are created for each of the data to be sent to the webservice, namely the byte array representing the image, the latitude and latitude ref, and the longitude and the longitude ref.

  The next step in this method is to send the data over to the webservice. This is done through the SOAP protocol by using a `SoapObject` and a

[4] https://github.com/mdinstuhl/Android_Camera_Demo

`SoapSerializationEnvelope` defined in the kSOAP2 [5] library. It is important to note that due to size restrictions on the data sent through SOAP, images are always taken to be the smallest resolution supported by the phone, that is 640x480px. The `SoapObject` requires as parameters the namespace of the webservice and the method name to be used. Since the webservice is implemented in C#.NET, the dotNet attribute of the `SoapSerializationEnvelope` was set to true. The `setOutputSoapObject` method was used to set the `SoapSerializationEnvelope` as a request.

```
SoapObject request = new SoapObject(NAMESPACE, METHOD_NAME);
SoapSerializationEnvelope envelope = new
SoapSerializationEnvelope(SoapEnvelope.VER11);

envelope.dotNet = true;
envelope.setOutputSoapObject(request);

HttpTransportSE androidHttpTransport = new HttpTransportSE(URL);
androidHttpTransport.call(SOAP_ACTION, envelope);

SoapPrimitive result=(SoapPrimitive)envelope.getResponse();
```

The two lines before the last of the sample code above show the sending of the SOAP request by using the `HttpTransportSE.call` method. The last line shows the explicit casting of the response received from the webservice to a `SoapPrimitive` object. This response is received in the form of a string delimited by ";" in the format *"location of user ; recognized image ; distance"*, where *location of user* is the location of a training image nearest to the location where the query image was taken, the *recognized image* is a match between the query image and a training image, and the *distance* is the approximated distance between the location of both images. It is important to note that if there was no matched image or no training images available the string would only contain one of the following strings accordingly:

- "No matched image";
- "No Training Image available".

However, if the location of the user is nearest to the matched image, the first and second parts of the string are equivalent. This is better explained in Section 4.2.2.1.1. If-statements are used to check the contents of the response string, and output the results to the user accordingly. If the response is in the format *"location of user ; recognized image ; distance"*, and the location of the user is different to the recognized image, the output to the users is as follows:

[5] Downloaded from http://code.google.com/p/ksoap2-android/source/browse/m2-repo/com/google/code/ksoap2-android/ksoap2-android-assembly/2.5.2/ksoap2-android-assembly-2.5.2-jar-with-dependencies.jar

You are at <location of user> and you are looking at <recognized image> which is approximately <distance> metres away.

If, on the other hand, the location of the user is equivalent to the recognized image, the output is as follows:

You are at <location of user>.

The results are communicated to the user by using the `TextToSpeech.speak` method which accepts the given strings as parameters.

- **onInit**
  This method is called after a new `TextToSpeech` object is initialized. If the initialisation was successful, the language of the `TextToSpeech` object is set, in this case to US English. If the language is not set successfully because the language is not available or not supported, the user is prompted to install any missing requirements from the market. Since the new `TextToSpeech` object is initialized on start-up of the application, the `onInit` method is used to give the user instructions by calling the `TextToSpeech.speak` method. Similarly, the latter method is called each time a voice prompt is required.

4.2 SERVER COMPONENT

The server component, implemented in the C#.NET 4.0 framework, is made up of two processes; namely the training process and the query process. This component was implemented in C# due to previous extensive experience with the framework, as opposed to C++ or C, which one might argue would be more fast for an application such as the one proposed. However, this system was developed keeping efficiency in mind, thus a trade-off was made to develop an efficient system within the given time restrictions. Also, C# enables easy development of webservices, such as the one implemented for the query process. OpenCV[6], a library of programming functions for real time computer vision, was used in the implementation of the server component. EmguCV[7] was used as a cross platform .NET wrapper to OpenCV. The Microsoft.SqlServer.Types[8] library was used when spatial data types, such as geometric and geographic data types, were needed.

4.2.1 TRAINING PROCESS

The training process involves the populating of the feature database, which is then used for object recognition. The training process can be run more than once, obviously to add new images. However, it has to be run at least once before a query can be executed successfully.

4.2.1.1 User Interface

The user interface is a very simple form which enables the user to select a number of images from the local machine. As can be seen in ***Figure 61***, the form is made up of two buttons and a progress bar. The top button opens a dialog box enabling the user to select the desired images,

[6] Downloaded from: http://opencv.willowgarage.com/wiki/
[7] Downloaded from: http://www.emgu.com/wiki/index.php/Main_Page
[8] http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.types.aspx

while the bottom button is used to truncate the database. As explained in the design, the user interface was implemented in a very simple manner because even though this process is not directly aimed for the use of visually impaired persons, the latter can run this process easily by using a screen reader.

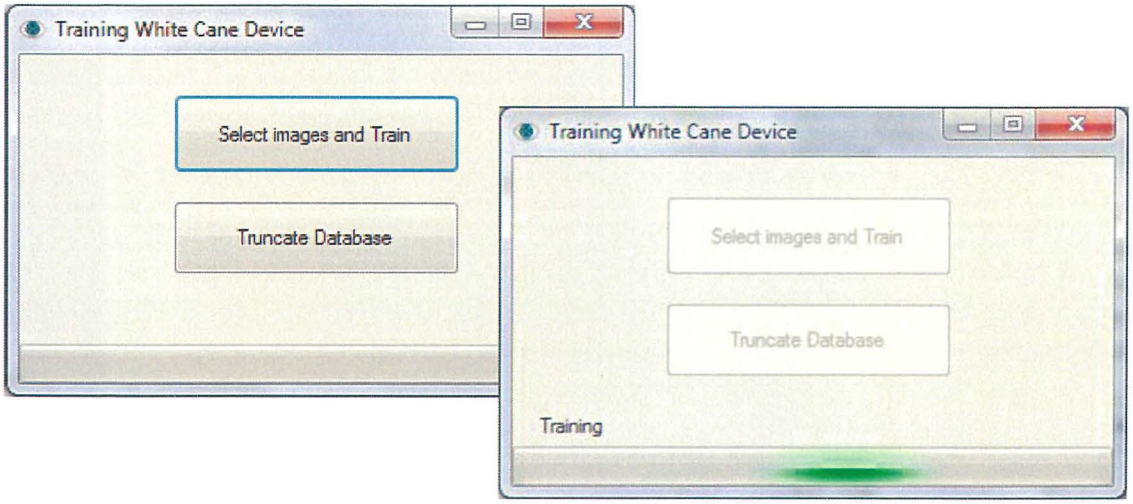


Figure 61: Training UI

The Truncate Database button is self-explanatory; all the data stored in the database is deleted. The Select images and Train button launches a series of processes. On button click, a background worker is implemented, which calls the methods required by the training process. Message boxes are shown upon completion of the processes of both buttons to confirm success of operation.

It is important to note that training images are added to the database of features successfully only if they are geo-tagged. The insertion of images which are not geo-tagged in the database would result in the GPS optimization in the query process becoming futile.

4.2.1.2 Backend

The Select images and Train button, as described above, is used as an entry point to a number of processes which are described below.

4.2.1.2.1 GPS extraction

The GPS coordinates are extracted from each image in turn. The coordinates are extracted in the format "DDD MM SS" and "N\0" where D, M and S represent degrees, minutes and seconds respectively, while N represents the orientation. To enable the building of a database containing only correct data, images without GPS coordinates are discarded.

4.2.1.2.2 Feature Extraction

The SIFT algorithm was chosen for the implementation of the feature extraction process. In Section 2.6.5, the evaluations led by [34] and [35] were presented. According to both papers' extensive evaluation, even though the selection of the ideal feature extraction algorithm depends

on the problem at hand, the SIFT algorithm performs best overall. In [35], SIFT performed best in the evaluations of scale, blur (in a textured scene), JPEG compression, illumination, and in the matching example. In [34], SIFT performed best in the evaluations of scale, rotation and blur (with a large radius). ASIFT was also considered since the evaluation in [25] gave good results, however no other evaluations which compare ASIFT to other feature extraction algorithms exist yet. Therefore SIFT was favored for having more comparative evaluations and the availability of supporting libraries.

An enquiry with Lowe, the author of the SIFT algorithm, confirmed that he did not run a complexity analysis on his algorithm. Since the SIFT algorithm processes images according to their resolution and content, the complexity of the algorithm as a whole cannot be calculated - changing some variables in the algorithm would result in a different complexity. Yet, some of the processes involved can be calculated separately. According to [42], the complexity of the scale space construction (explained in Section 2.4.2.1) for 640x480 monochrome images is as follows. The calculations assume three pyramid levels and three scales per level and represent the minimum operations for SIFT.

$$O(N \,.M \,.P^2) + O(N \,.M)$$

Where

N is the image width;

M is the image height;

P is the (symmetric) filter size, e.g. 9x9.

The first term is the result of the convolution while the second term is the result of the resampling. Accordingly, for a 640x480 image, building the scale space takes about 100,000,000 operations.

Consider the application for this thesis; a visually challenged person is required to take a picture to be used for object recognition. The first image transformation which comes to mind is blur. The visually challenged person cannot focus properly, even though the implemented mobile application has auto focus. However, users using a mobile phone with a different operating system (which would require a different client implementation) might not support such a feature. Thus blur was an important image transformation which required to be considered. Other important image transformations which required consideration are scale and viewpoint change. A user might take an image of a particular object from various distances and viewpoints. One might also say that illumination is relevant in the context of this application since various times of the day (and night) result in image transformations. Time is also important for the application in question, since the user requires real time results.

Keeping in mind the mentioned requirements as the most important, but also giving the other transformations some consideration, SIFT results the best choice for this implementation. Even though SIFT does not outperform the other algorithms in all image transformations, it gives the best trade-off between the performance on various image transformations. This gives the best overall feature extractor for the purpose of this implementation.

Various open source implementations of the SIFT algorithm exist, however, due to previous experience using C#.NET, a C# version of the SIFT algorithm, ported from a C++ version was implemented in this thesis[9]. This implementation is very similar to the SIFT algorithm as described in Section 2.4.2.1, with the only difference that the scale space is computed in parallel. This modification, which allows faster implementation of the algorithm, was done based on the [43] and [44]papers.

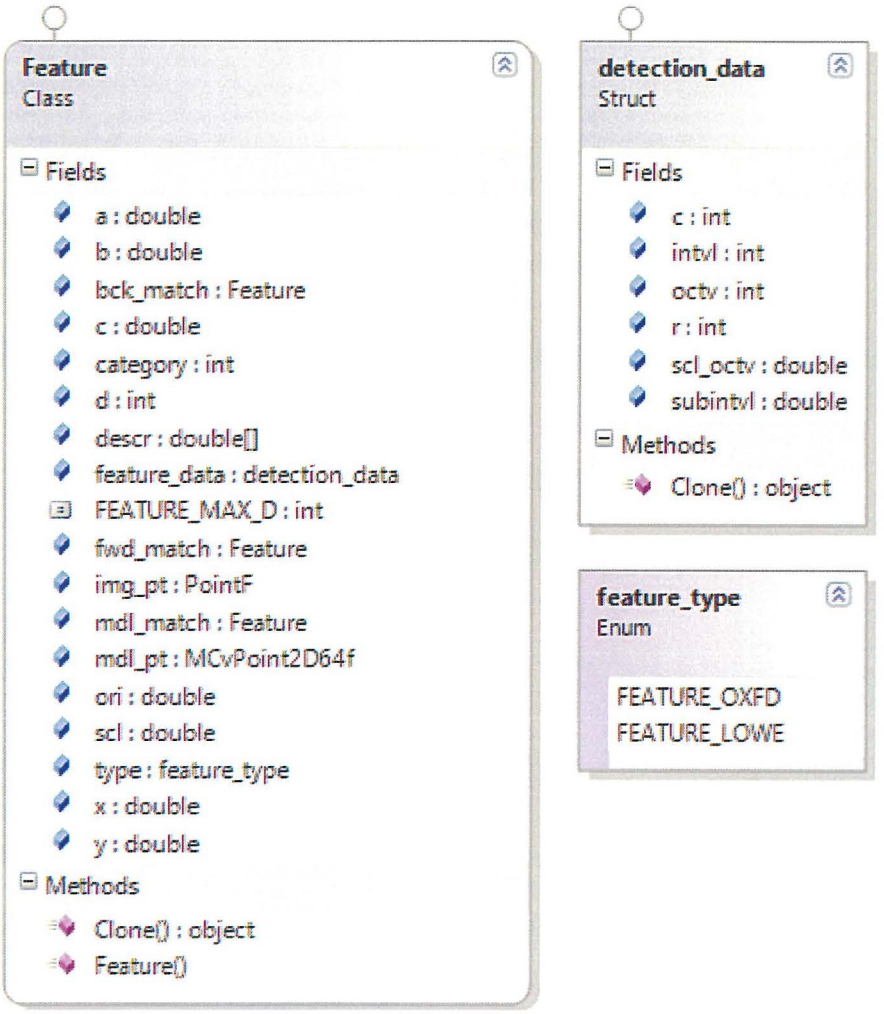


Figure 62: UML class diagrams for Feature class

[9] Downloaded from https://sites.google.com/site/btabibian/projects/3d-reconstruction/code on 22 February 2011

The SIFT algorithm implemented in this thesis accepts a greyscale image and returns a list of `Feature` objects for each training image. ***Figure 62*** shows the class diagram for the Feature class. It is important to note that we use features as described by Lowe in [18], and that we will only be using the feature descriptor, represented by an array of double values, for further processes.

4.2.1.2.3 Saving to database

Up to this point, the backend of the training process has extracted the GPS coordinates from a training image, and computed a list of features from the same image. This data is added to a `SerializableFeatureList` object, along with the respective GPS coordinates and image name. ***Figure 63*** shows the class attributes. Thus, we now have a `SerializableFeatureList` object for each training image selected by the user.

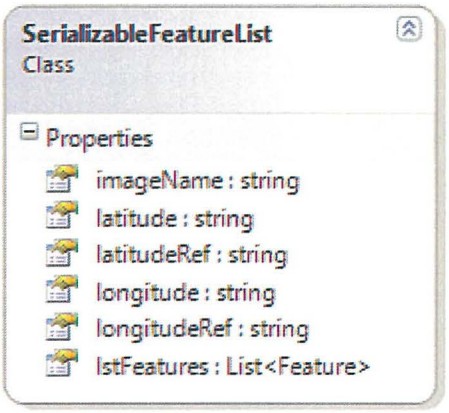


Figure 63: SerializableFeatureList UML class diagram

The next step is to actually save the features to database. The following process is repeated for each feature extracted, for all training images selected by the user:

The feature descriptor is extracted from each `Feature` object, and transformed into a string. This is done simply by implementing string concatenation with blank space as a delimiter, for all 128 double values stored in the array representing the descriptor. Then, the latitude and longitude strings are transformed in an array of double values. This array is in turn used along with the latitudeRef and longitudeRef to transform the coordinates to decimal format as explained in Section 2.1. An `SqlGeometry` object is used so store this GPS data as follows:

```
SqlGeometry geo = SqlGeometry.Point(decimalLatitude, decimalLongitude, 4326);
```

Where 4326 is the SRID for the WGS 84 ellipsoidal Earth model; that is, the reference coordinate system used by the GPS. An `SqlCommand` object is then executed to save the feature to the database, along with its respective coordinates, the descriptor, and the name of the training image from which the feature was extracted. The `CommandText` attribute of the `SqlCommand` object is set to the following text:

```
"INSERT INTO dbo.Features (ImageName, Descriptor, Gps) VALUES ('" + imageName + "', '"
+ descrString + "' , '" + geo + "')"
```

This will result in the appending to the database a number of features derived from a single image, with different descriptors and identical GPS coordinates and image name.

4.2.2 *Query Process*

The query process is started by a call from the client to the webservice, which acts as an interface to the server application. The webservice accepts a byte array representing the query image, and four strings representing GPS coordinates of the image at hand. The latter strings can be null values if the query image at hand did not have GPS coordinates. When the query process is complete, the webservice returns a string representing the result to the client.

The query process starts with the re-building of the image from the byte array. The resulting bitmap image is then transformed into greyscale and passed to the SIFT algorithm. Feature extraction is then computed as explained in Section 4.2.1.2.2. The descriptor of each feature is then used to build a matrix of float values. Consider the example of 2345 features which were extracted from the query image. The matrix in this case would consist of 2345 rows, one for each feature, and 128 columns, one for each value in the feature descriptor.

The next step is to extract relevant features from the feature database. This is done by calling a method, and passing the query image GPS coordinates as well as a pre-set radius value as parameters. This radius value is used to define a specific area around the location the query image was taken. The features extracted from training images taken within this area are loaded from the database. This radius value makes the object recognition process more efficient and reliable [18], compared to loading the entire database, and subsequently having to compare all the features. The default value for the radius is set to 50m. The features are loaded from the database by calling a stored procedure, which is described in the following sections. For query images without GPS coordinates, all the features within the database are loaded.

Each loaded feature is stored in an `ExtractedFeature` object containing the unique feature id (as stored in the database), the image name from which the feature was extracted, the descriptor of the feature, as well as the distance between the location where the query image was taken and the location the training image in question was taken. This distance must fall within the defined radius. ***Figure 64*** depicts the UML class diagram for the `ExtractedFeature` class. This method results in a list of `ExtractedFeature` objects, one for each feature extracted from the database. At this point, the size of the mentioned list is checked. If the list contains no features, then there are no training images available, and this result is returned to the client.

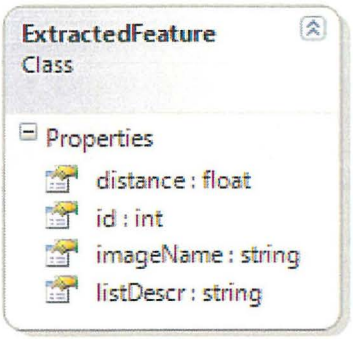


Figure 64: UML class diagram of the ExtractedFeature class

The list of `ExtractedFeature` objects is then used to build the dataset matrix of float values, similar to the one created using the query features. Thus, if 10,000 features were loaded from the database, the matrix will have 10,000 rows and 128 columns. During the building of the matrix, a hashtable mapping each row from the matrix (representing a single feature) to its respective image name and distance is created. The value of the hashtable is represented by an array of string. The building of this hashtable is done to aid in the matching process.

NameMappingTbl	
Key	**Value**
0	[University entrance, 57.53098]
1	[University entrance, 57.53098]
...	...
4500	[University canteen, 14.45219]
4501	[University canteen, 14.45219]
...	...
x	[Image name, distance from query image]

Figure 65: Table showing NameMappingTbl hastable contents

4.2.2.1.1 Object recognition

The system proposed by Lowe and Muja [32] was used for the implementation of the object recognition process. Given a dataset and the desired degree of precision, the proposed system automatically determines the best nearest neighbor algorithm and the respective parameter values. According to experiments led by [35], nearest neighbor distance ratio matching approaches give the best performance when matching descriptors such as the SIFT descriptor. The system developed by Lowe and Muja uses a "cross-validation approach to identify the best algorithm and to search for the optimal parameter values to minimize the predicted search cost of future queries" [32]. The system also allows the user to fine-tune it according to the requirements at hand; such as preferring a less than optimal query time in exchange for reduced memory usage, a smaller data structure build-time, or less time spent on parameter selection. In experiments the authors led, the latter show that it is possible to obtain a speed-up by a factor of 1,000 times when compared to linear search, while still identifying 95% of the correct nearest neighbors.

FLANN[10] is a library containing the system proposed by [32] and a collection of algorithms which were found to work best in nearest neighbor search problems in high dimensional spaces. OpenCV provides an interface to the FLANN libray, which we used through EMGU.

The object recognition process starts with the building of a kd-tree index using the dataset matrix. This is done simple by calling a library routine. Next, the nearest neighbors for each feature extracted from the query image are computed using the index just created. The method performing this computation accepts five parameters as follows:

```
KdTreeIndex.KnnSearch(Query, Indices, SqDist, KNN, CHECKS);
```

Where;

- **KdTreeIndex** is the index just built;
- **Query** is the matrix of float values (size is: *number of features* x 128) built using the descriptors of features extracted from the query image;
- **Indices** is a matrix of integer values (size is: *number of features loaded from database* x *number of nearest neighbors specified*) providing mapping to the respective feature in the dataset matrix;
- **SqDist** is a matrix of float values (size equivalent to Indices matrix) representing the Euclidean distance between a query feature and a nearest neighbor specified in the Indices matrix;
- **KNN** is the specified number of nearest neighbors required. The default value for this implementation is set to 2;
- **CHECKS** is the maximum number of leaves to visit when searching for neighbors. A higher value for this parameter would result in better search precision, but also take more time. The default value for this implementation is 200, as specified in [18].

This method thus populates two matrices, the Indices matrix and the SqDist matrix. These matrices indicate the first two nearest neighbors to each feature extracted from the query image, and their respective distances from the query feature in question.

The final and crucial step left is the computation of the results. Initially this was done using a weighting system, however the final implementation was deemed to be better. The initial implementation made use of the distances between the query image and the training images and used them as weights on the number of matched features to compute a score. The score was calculated by dividing the number of features from a training image which matched the query image by the distance between both images. The training image with the highest score was taken to be the recognized image. The weighting was based on the assumption that the further a training image is from the query image, the less probable it is that the training image matches the query image in question. Consider having 100 features each from two training images A and B which match a query image X. If the distance between A and X is smaller than between B and X, then the matching image is taken to be A.

[10] http://www.cs.ubc.ca/~mariusm/index.php/FLANN/FLANN

This initial implementation raised some problems. Imagine having two training images A and B and a query image X. If one takes an image of B from very near A, then the weighting system would give an incorrect result, since the correct image, that is B, would get a lower score than A. This is because the distance weighting would overcome the number of matched features. Thus, we considered using the image orientation, and match the query image only with images facing that orientation. This option was however discarded in favor of a different implementation which gives a different set of results.

The results in the final implementation contain the following three things in the format *"location of user ; recognized image ; distance":*

- An approximation to the location of the user when the query image was taken. This result's accuracy depends on the content of the database. The approximation is calculated on the training image whose GPS coordinates are nearest to the coordinates of the query image. Thus, if the database contains more training images of the particular place in question, the result is more accurate;
- The recognized object within the query image;
- The approximated distance between the current location of the user and the recognized object. This data has already been calculated when the features were loaded from the database.

The *recognized image* result is computed by calculating the distance ratio between the first and the second nearest neighbor for each feature of the query image, using the Indices and SqDist matrices. This is done as explained in [18]; features whose distance ratio (distance of first nearest neighbor / distance of second nearest neighbor) is larger than 0.8 are discarded. This "eliminates 90% of the false matches while discarding less than 5% of the correct matches" [18]. This measure performs well as correct matches require having a nearest neighbor significantly closer than the closest incorrect match for reliable matching. False matches, on the other hand, will likely have a number of other false matches within similar distances due to the high dimensionality of feature space. Using the NameMatchingTbl hashtable described earlier, a new hashtable is built. This time, the NameCount hashtable contains the following data:

NameCount	
Key	**Value**
University Entrance	[93, 57.53098]
University canteen	[69, 14.45219]
...	...
Training Image name	[Number of correct features, distance from query image]

Figure 66: Table showing NameCount hastable contents

This hash table thus contains the name of the training image from where features were extracted, as well as the number of matched features, and the distance between the query image and the training image in question. Thus, if the features from 5 images were loaded from the database, and if at least 1 feature from each image correctly matched a query feature, the NameCount hashtable would contain five key values containing the names of the five training

images, along with the respective count and distance values. Here we are assuming that each training image has a unique name.

Each key-value pair in the hashtable is then used to create a `FeatureName` object. ***Figure 67*** shows the UML class diagram of the latter. All `FeatureName` objects are used to create a list. This list is then counted; if it is empty or the maximum number of matched features is less than a given threshold, then the client is given a "No matched image" result. This threshold will be determined after evaluating the system. Otherwise, the list is sorted according to the image name having the maximum number of matched images. Thus, continuing with the example portrayed in ***Figure 66***, 'University Entrance' would rank first, followed by 'University canteen'. The image ranking first is the recognized object within the query image.

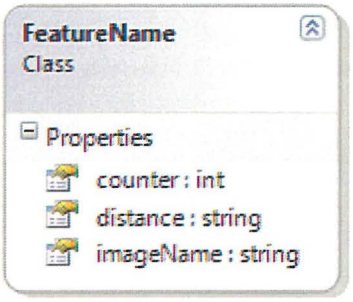


Figure 67: UML class diagram for the FeatureName class

The list is sorted again, this time according to the distance attribute. Continuing with the example portrayed in ***Figure 66***, 'University canteen would rank first, followed by 'University Entrance. The image ranking first is the training image nearest to the location where the query image was taken, thus giving the approximation as to the location of the user when he/she took the query image. This is the first part of the results. The third and last part of the results is the distance attribute of the image which ranked first; in the case of 'University canteen' would be '14.45219m'.

The sorting of the list according to the distance attribute is done as following. The sorting of the list according to the count attribute is done similarly.

```
NameCountLst.Sort(delegate(FeatureName f1, FeatureName f2)
{
    return (f1.distance.CompareTo(f2.distance));
});

result = NameCountLst[0].imageName;
```

It is important to note that the recognized image and the approximated location might coincide. This occurs when the recognized image is also the training image nearest to the user's location.

Consider having two training images taken from different, but near, locations A and B. If a user takes a query image of B from A, then the result containing the matched image will contain A and the approximated location will contain B. If, on the other hand, the user takes an image of B, from very near B itself, both results will be B.

Upon completion of the computation of the results, the latter are returned to the client in String form. Of course, one must note that having a query image without GPS coordinates, the distance between the query image and the training images cannot be computed. Therefore, this last mentioned process is skipped, and only the recognized object within the query image is output as a result.

4.3 Database Component

The database component was implemented using Microsoft SQL Server 2008 R2. The feature database is made up of a single table populated by the training process as described in Section 4.2.1.2.3. Simply put, the database contains the respective data of all features extracted from all training images, namely the image name, the descriptor and the GPS coordinates.

Column Name	Data Type
Id (Primary Key)	int
ImageName	nvarchar(100)
Descriptor	nvarchar(MAX)
GPS	geometry

Figure 68: Database Schema

As described earlier, both the query and the training processes interact with the database. The training process simply appends rows of data to the table. The query process, on the other hand, passes the GPS coordinates of the query image as a parameter to a stored procedure. The stored procedure is a query which enables the loading of features which were extracted from images taken within a particular radius from where the query image was taken.

The stored procedure calculates the distance between the query image GPS coordinates passed as a parameter and the respective GPS coordinates of the features by implementing the Haversine formula explained in Section 2.1. The implementation is as follows:

```
(6371000 * (2*(asin (dbo.MinArgument(1,sqrt((Power (sin((RADIANS(@latitude)
- RADIANS(Gps.STX))/2),2)) + cos(RADIANS(Gps.STX)) * cos
(RADIANS(@latitude)) * (Power(sin((RADIANS(@longitude) -
RADIANS(Gps.STY))/2), 2)))))) )
```

Where *latitude* and *longitude* are the respective GPS coordinates of the query image and *Gps.STX* and *Gps.STY* are the latitude and longitude GPS coordinates of the feature at hand. *MinArgument* is a function which returns the minimum of two values, while the *Radians* function converts the degree coordinate to radians. The value of the Earth's radius is taken in meters in such a way

that the resulting distances are all in meters. All features whose resultant distance is less than the given radius value are passed back to the query process.

If the query image in question does not have any GPS coordinates, instead of calling the stored procedure, a simple query is executed which loads all features within the database.

4.4 Conclusion

In this chapter we have defined any implementation choices made for each of the components of the system and the main processes. Any relevant approaches to solve the problem at hand were also explored, including libraries used. We described the various user interfaces of the system. The implementation for the training and query processes was described in detail, as well as for minor processes such as feature extraction, GPS extraction and saving to database.

Chapter 5

EXPERIMENTS AND EVALUATION

In this chapter we run a series of tests and experiments to determine any variations in the results when system variables are changed. Such tests also help us determine and quantify the contribution of various system features to the end result. To aid the experimentation process, a testing module was added to the server-side, since testing the system thoroughly would be very cumbersome through the default client-server approach. It is important to note that this is not an evaluation of the SIFT feature extraction algorithm per se, since innumerable evaluations already exist in literature. Rather, this evaluation focuses on the implementation of the SIFT algorithm within the developed system in the context of an object recognition problem.

5.1 EXPERIMENTAL SETUP

To enable easy experimentation, a testing module was added to act as an interface to the query process described in Section 4.2.2, to replace the default webservice used by the client. This interface is a simple form, as shown in ***Figure 69***, which enables the user to select a number of query images at one go. The form also enables the user to modify the radius value as described in Section 3.1.3. To aid the data gathering, required information obtained while computing the results are written to a text file.

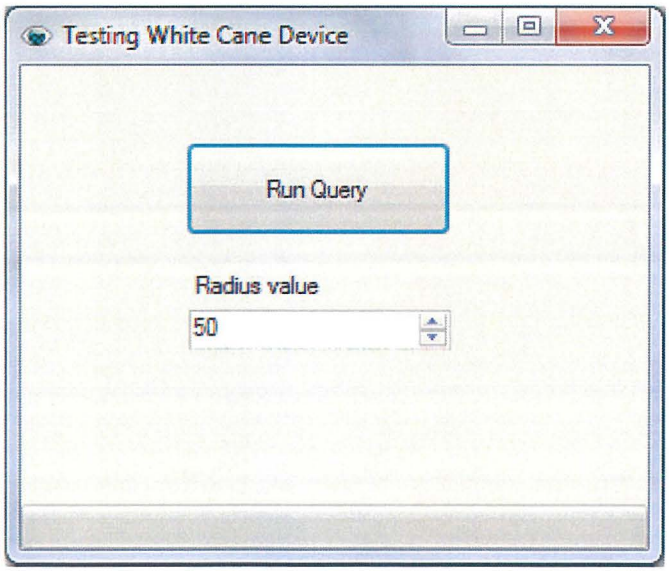


Figure 69: Testing UI

5.2 TESTING DATASETS

5.2.1 TSG-40-HTC DATASET

The TSG-40-HTC[11] dataset is made up of 246 JPEG real-world images taken in the inner city area of Graz, Austria. The images were taken using an HTC Touch Cruise camera enabled mobile phone. The images were taken at 3 megapixel resolution, from different viewpoints and under different weather and illumination conditions. Some images also contain occlusion. A spreadsheet file accompanying the dataset contains information regarding each image, such as the GPS coordinates, the weather conditions and the viewpoint.

The dataset contains 6 images per object taken from viewpoints 1,2,3,4,6,8 as shown in ***Figure 70***. The images were captured using neither flash nor zoom. The GPS coordinates were obtained using a GPS device in such a way that the GPS data corresponded to a real world scenario. The pictures were taken at different times of the day, namely:

- morning (8 - 10 am)
- noon (10 am - 2 pm)
- afternoon (2 - 5 pm)

and under different weather conditions, namely sunshine and cloudiness.

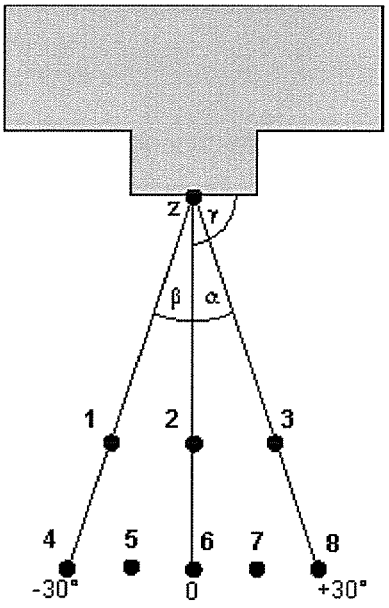


Figure 70: Dataset viewpoints. Z is the central point of the building (the middle of façade), Gamma is of 90 degrees, Alpha and Beta are of 30 and -30 degrees respectively. Image from [47].

[11] Downloaded from http://dib.joanneum.at/cape/TSG-40/index.php?page=download

Since the TSG-40-HTC images were not geo-tagged, but the GPS coordinates were available, we developed a program to add the GPS coordinates to the dataset images as properties. Simply, this program read the respective coordinates of the image in question from a spreadsheet, and created a new image with the added GPS properties. Also, to reduce the number of features in the database, the images were all scaled to 853x640. This enables faster computation of results, and yet does not necessarily hinder the matching process, as evaluated in [38].

Figure 71: Sample images of the TSG-40-HTC database. Images (from left to right) are taken from viewpoints 1, 2 and 3 respectively. Image from [46].

5.2.2 *UoM Dataset*

The UoM dataset is a number of JPEG images depicting various buildings around the University of Malta, Msida, Mdina and Sliema. The images were taken under various illumination and weather conditions, and are all geo-tagged. An HTC Desire mobile phone with a 384x640 resolution was used to capture this dataset. The images were taken from random distances and viewpoints, in order to mimic the action of a real-life user of the proposed system. This dataset also has some images containing occlusion.

Figure 72: Sample images of the UoM dataset.

5.2.3 *Training*

We started off the testing process by training the system. This was done using the training process as described in Section 4.2.1. For the training images, an image of each object in the TSG-40-HTC dataset was used. The images chosen were all taken from viewpoint 6, since it best views the entire building façade and enables best matching. 13 images from the UoM dataset were also used as training images. The training process resulted in around 97,500 features being stored in the database.

5.3 Evaluation Metrics

Receiver Operating Characteristics (ROC) and Recall-Precision are popular metrics in literature, sometimes even used interchangeably. Both metrics can be used to show that we want to increase the number of correct positive features, while decreasing false positives. Yet, as one can observe in [45], ROC are more suited to evaluate classifiers, since the false detection rate is well-defined. Recall-Precision, on the other hand, are better-suited to evaluate detectors (such as in the case of this thesis), since the number of false positives relative to the number of all detections can be correctly expressed by 1-precision, even though the total number of negatives cannot be determined.

This evaluation is similar to the one in [35], where the performance of the system is measured on a keypoint matching problem, that is, having a feature extracted from a query image (the query feature), find all the matches of the query feature from the feature database. Since this is clearly a detection problem, rather than a classification one, Recall-Precision are used for this evaluation. ***Equation 3*** in Section 2.6.2 shows the Recall and 1-Precision equations.

The graphs of Recall against 1-Precision are constructed as follows. First, the required features are loaded from the database according to the given radius. These features are extracted from various images, along with features from the training image of the object portrayed in the query image. The first and second nearest neighbors for each query feature are then found. Nearest neighbor distance ratio matching is then implemented. The features which pass the ratio threshold are the total number of matches. These features can belong to various images, amongst which the correct training image with respect to the query image in question. The features which belong to the correct training image are termed *correct positives,* while those which belong to other training images are termed *false positives.* The number of features extracted from the correct training image make up the total number of positives.

5.4 Experimental Results

In this section we present and discuss the experimental results of the evaluation. We explain the purpose of the experiments, any variables which were modified, and the outcomes of the experiments. Assume that the same random sample of 20 JPEG images from the TSG-40-HTC dataset was used as test data unless otherwise specified, and that no training image was used as a query image. Also assume that the radius value is taken at 50m unless otherwise specified. One should note that this radius returns most of the training images from the TSG-40-HTC dataset. The time taken is measured from before the loading of the image, to the computation of the final result. The resulting matched images should be assumed to be correct unless otherwise stated.

5.4.1 *Number of checks for Nearest Neighbor approximation*

This experiment involves varying the number of checks made for the approximation of the nearest neighbor with respect to the query feature in question. In [18], Lowe defines this value to be 200 for the Best-Bin-First algorithm. The purpose of this experiment is to determine the varying of the accuracy of the results, as well as the varying in the time taken.

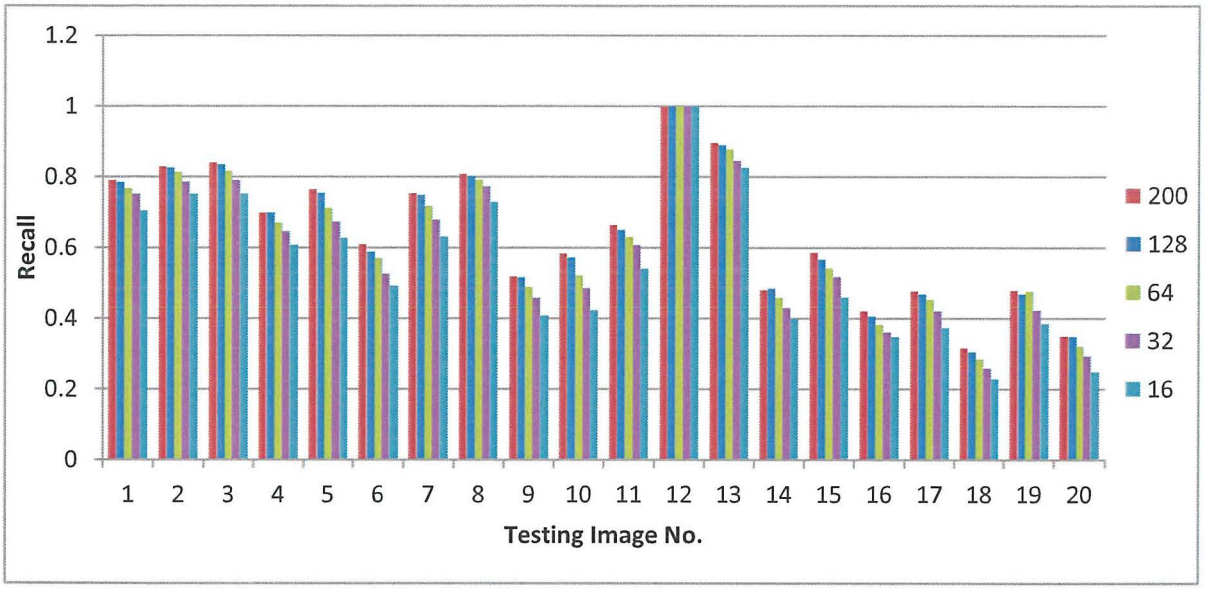


Figure 74: Graph showing recall for Number of checks for Nearest Neighbor

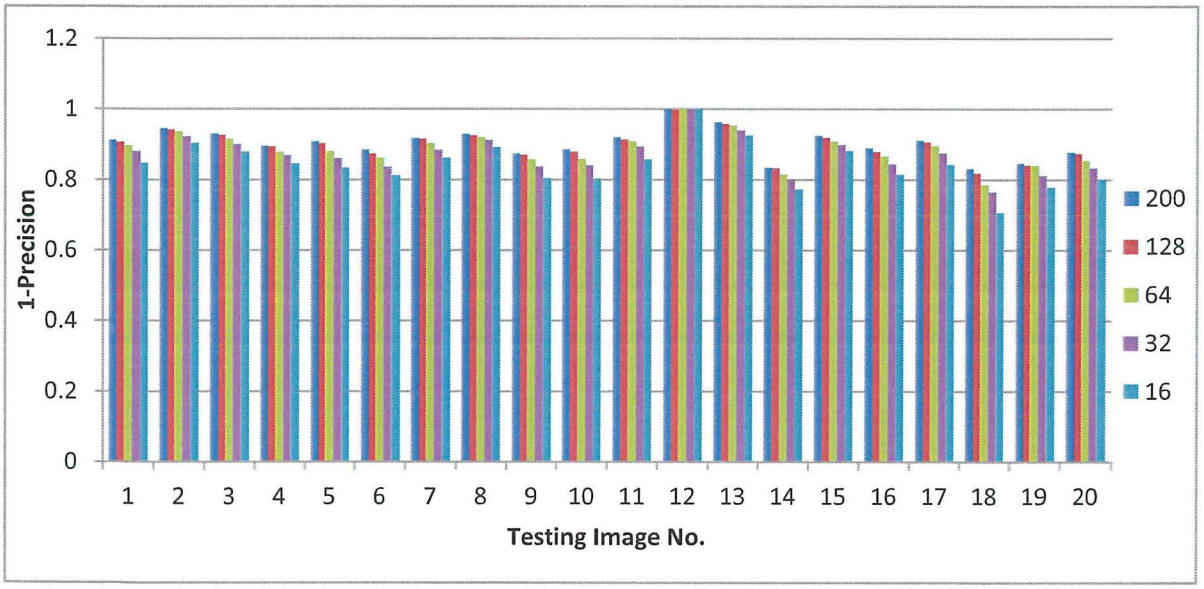


Figure 73: A graph showing 1-Precision for Number of checks for Nearest Neighbor

The results in both ***Figure 74*** and ***Figure 73*** show that precision and recall both diminish as the number of checks is reduced. The recall has a larger range of values for the different images

while the 1-precision is similar for all images. The 1-precision values are mostly above 0.8, showing that the check value used does not have such an impact on the resulting false positive features. The recall values are somewhat more affected, as the number of correct features is reduced a notch for each descending tested value of checks. The query images in these tests all gave the respective training images as the resulting matched image. Thus we can see that even though the reducing of the number of checks made to find the nearest neighbour resulted in slightly worse values for the recall and 1-precision, the end results were more than satisfactory for all the number of checks.

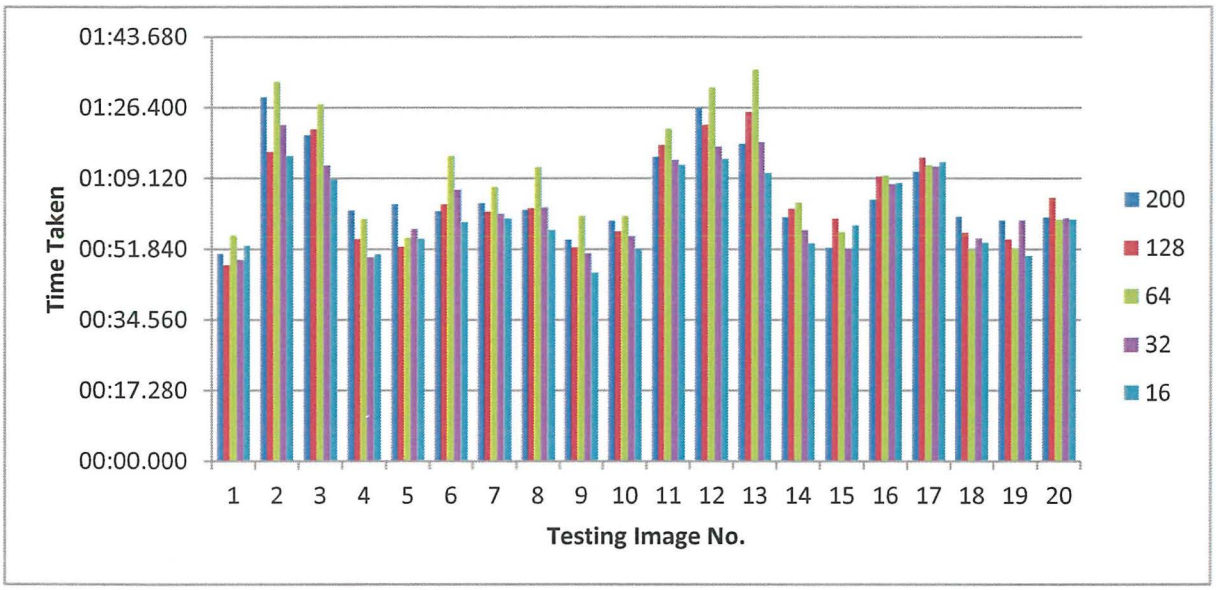


Figure 75: A graph showing the Time Taken for Number of checks for Nearest Neighbor

As can be seen in ***Figure 75***, the time taken to compute the results varies and is not always proportional to the number of checks being made. This might be due to some background process running on the machine during the testing, even though utmost care was taken to prevent this. Various tests all resulted in similar time patterns.

5.4.2 THRESHOLD FOR NUMBER OF FALSE POSITIVES HAVING NO TRAINING IMAGE

This experiment is done to determine a threshold to set on the number of allowed "correct positives" for a training image to be defined a match. In fact, when there is no training image to the query in question, there can be no correct positives at all. These features are simply features which pass the ratio threshold and belong to the training image with the highest number of matches. This experiment is done by comparing the number of correct positive features for a set of testing images; once having their respective training images in the database, and once not (by removing the respective training images from the feature database). This threshold is required so that if a user takes a query image of a building of which there is no training image in the database, then the system does not return a wrong match.

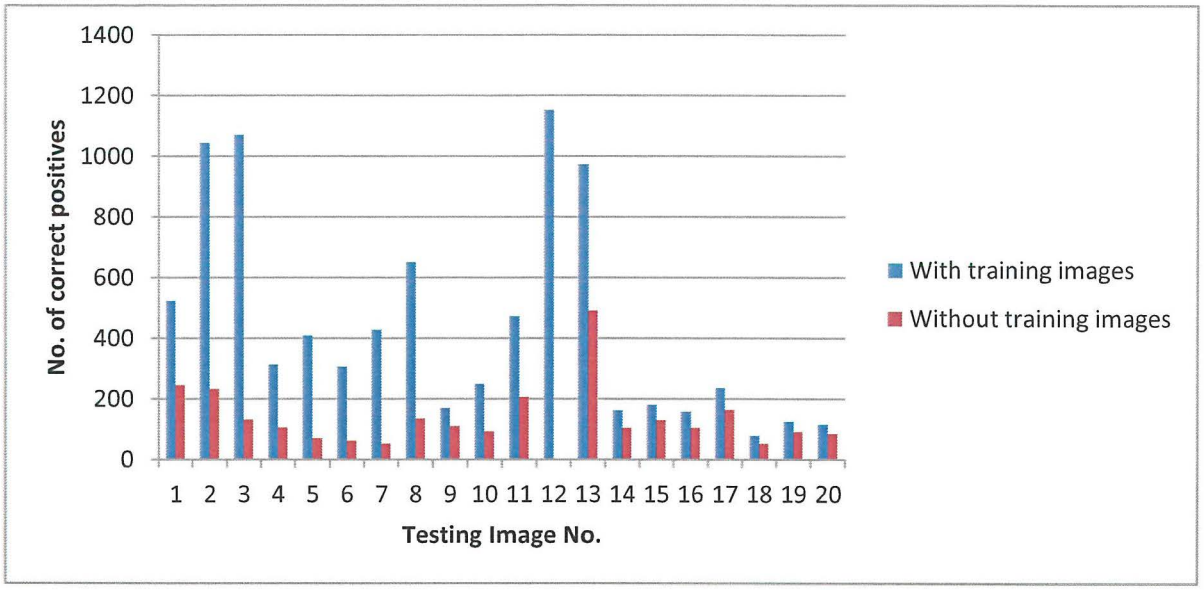


Figure 76: Graph showing the number of correct features for test images with and without respective training images

The results in ***Figure 76*** show that for most images, the number of correct positive features is much larger when the database contains a training image matching the given query image, than for the opposite case. However, the large range of values for different images indicates that setting a threshold would not be efficient in determining images without a training image. Thus, a scenario where the query image does not have a respective training image in the database would result in the image most similar to the query image to be defined a match. Correct positive features in such a scenario are non-distinctive features, probably derived from clutter, which can be matched to various images.

5.4.3 SIFT vs. SIFT+GPS

In this experiment we compare the results obtained using the SIFT algorithm as defined by Lowe in [18], to the proposed implementation which exploits GPS coordinates to reduce the search space. For the traditional SIFT approach, testing images were used without any GPS tags in such a way that all the features from the feature database are loaded and used for the object recognition process.

Consider the following scenario: having two locations A and B near each other, the user takes an image of B standing nearest to A. If the database contains the training images matching both locations, then the result should contain the recognized object within the image, that is B, and the training image nearest to the location the query image was taken, that is A. One must keep in mind that the more training images the database contains of a particular location, the more the latter function can be accurate.

Testing images for both SIFT and SIFT+GPS were selected from the UoM dataset, since it contains scenarios such as the one described, and the results can be known a priori. Also, the UoM dataset contains images from sparse locations, thus the 50m radius loads only a small number of training images from the database. This is in contrast to the TSG-40-HTC dataset,

where most images are concentrated in a small area. The testing was done using 9 images of various buildings captured from different perspectives.

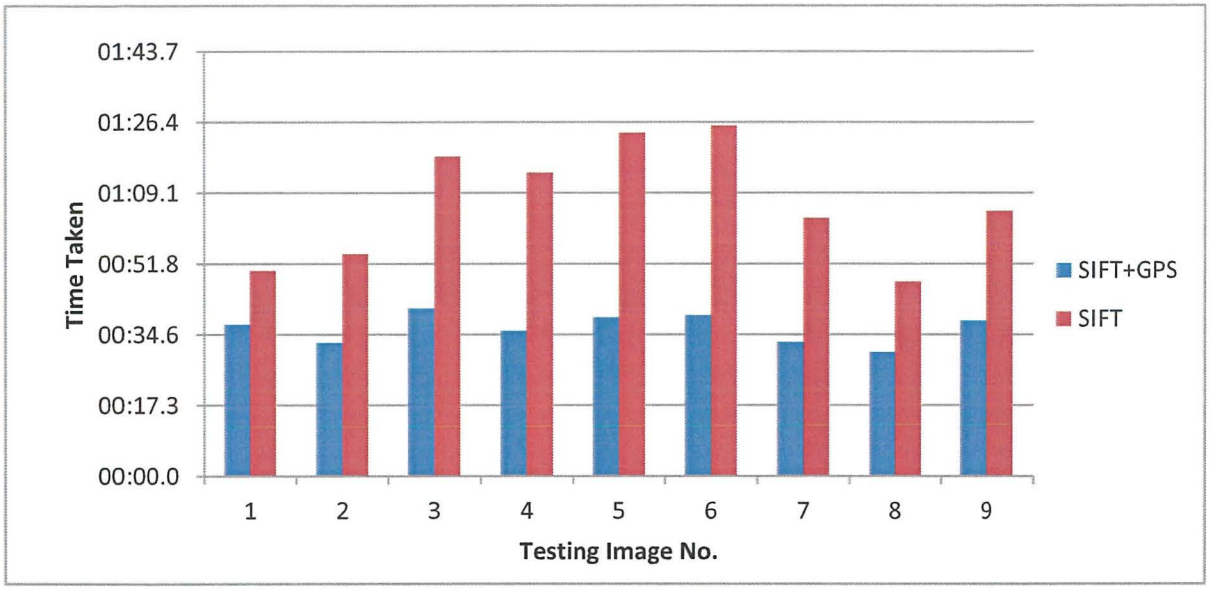


Figure 77: Graph showing the difference in Time taken for SIFT with and without GPS. Images 1,2 and 8 returned incorrect results for SIFT without GPS.

According to the results shown in ***Figure 77*** the average time to calculate the results for SIFT is 01:07 minutes while for SIFT + GPS is 00:36 minutes. This is nearly half the time taken by using SIFT without the GPS optimization. This discrepancy in the time is mainly due to the large difference in the amount of features loaded from the database. Though the loading itself does not make such a difference in the time taken, the computation for the nearest neighbors takes much longer. Also, without the GPS optimization, there is a larger probability that the query image is matched with the incorrect training image, as happened for images 1, 2 and 8.

Images 5 and 8 returned the current location of the user to be different than the location of the object recognized within the query image. The results were as follows:

Query Image	Time Taken	You are looking at:	You are at:	Approx. Distance from matched image
lecture building q.jpg	00:38.8	lecture building	computer building	26.88
library q.jpg	00:30.3	library	canteen	28.47

As explained, this means that, having a query image taken at location A, and two training image sat locations B and C, A was nearer to B, even though the query image depicted the object at C. For the "lecture building q" query image, the "computer building" training image is 26.09m away while for the "library q" query image, the canteen training image is 11.25m away. These training

images, having their locations nearer to the location of the query image than the locations of the matched images, were given as the "you are at" results.

5.4.4 *SIFT vs. PSIFT*

This experiment involves evaluating the implementation approach of the SIFT algorithm. As stated in Section 4.2.1.2.2, the proposed system implements a parallel version of SIFT (PSIFT), where the scale space is computed in parallel. The aim of this experiment is to evaluate any differences in time taken to compute the results between the default and the parallel implementation of the SIFT algorithm. This is done simply by passing a boolean variable to the SIFT feature extraction method, and the method is implemented accordingly.

Figure 78: Graph showing the time taken for SIFT vs. PSIFT

The results in ***Figure 78*** show that PSIFT is generally faster than SIFT. Some images prove the opposite; however this is probably due to the particular content of the image.

5.4.5 *Multiple training images*

The Nearest Neighbor distance ratio matching, explained in Section 4.2.2.1.1, requires the first and second nearest neighbors of a query feature to have a distance ratio larger than 0.8, as defined in [18]. Thus, the second nearest neighbor requires to be from a training image other than the training image depicting the object within the query image. For example, having a query image A, its respective training image B, and a different training image C, the first nearest neighbor should be from B while the second should be from C. Having multiple training images depicting the same object therefore might raise some problems. This is because if the first and second nearest neighbor to the query feature are taken to be the correctly matching features from two distinct and correct training images, then the distance ratio will be smaller than 0.8, as both features can be correct matches to the query feature. This phenomenon results in the correctly-matching features being discarded.

This experiment is led with the aim of evaluating any difference in the number of correct positive features with, and without, multiple training images. For the multiple training images part of the experiment, 6 images from the UoM dataset depicting 3 different buildings are added to the feature database. The testing was done using 9 images from the UoM dataset.

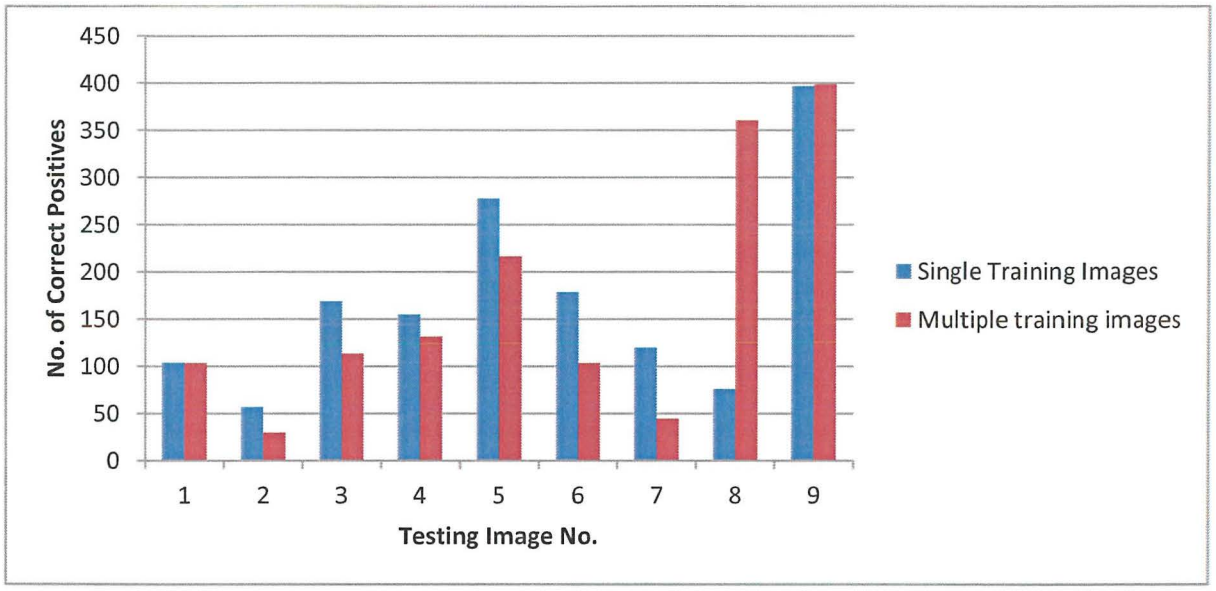


Figure 79: Graph showing the number of correct positives for a query image having single and multiple training images respectively. Images 3, 7 and 8 returned one of the new training images as result for the test using multiple training images.

Generally, having single training images results in a higher number of correct positive features. Yet, some images such as image 8 happened to match much better with the new training image, and resulted in a much higher number of correct positives. One should note that images 3 and 7 shown in ***Figure 79*** also returned a new training image as the matched image.

5.5 Client Evaluation

The user evaluation was done with the kind help of Mr. Michael Micallef and Mr. Stanley Debono from the Foundation for IT Accessibility (FITA). Micallef, who is visually challenged, was introduced to the White Cane Device and gave important feedback on its usability. Debono, who works on a day to day basis with visually challenged persons, also gave important suggestions. White Cane Device was considered to be of great aid to persons with visual impairments, and quite easy to use and understand with some basic instructions. Micallef suggested that instead of just returning the image matched with the query image, the system should also return the current location of the user if the latter is different than the matched image. This was implemented accordingly. Debono voiced a concern regarding having multiple training image of a particular object; namely if a query image matches more than one training image, what result is returned. The evaluation in Section 5.4.5 shows such a scenario; the end result being the image which has the largest number of matched features with the query image. Both Debono and Micallef agreed that the addition of further information with each training image would be much more useful to the end user. Such information could include the accessibility of the building in

question. Being a prototype, such information was not added to this implementation, however it was added to the Future Work section.

5.6 Conclusion

The various experiments shed light on the results achieved by changing properties within the SIFT algorithm. The system was tested using different values for the number of checks for the approximation of the nearest neighbor. The results confirmed that a higher number of checks results in higher precision and recall values. The timing results for the latter test were inconclusive. When query images do not have their matching training images, the training image most similar to the testing image is defined as the matching image, since the implementation of a threshold would not be effective. The exploiting of the GPS coordinates resulted in a great improvement on the time taken to compute the results; with SIFT+GPS taking nearly half the time taken by SIFT only. This optimization also enabled the user to be given more detailed results regarding his/her whereabouts. The PSIFT implementation resulted to be generally faster than the classic SIFT implementation. When tested with multiple training images, having more than one training image for a particular query image resulted in a smaller number of correct positive features, however the tests always returned the correct training image as the matching image. The use of datasets containing images taken from various perspectives also shows that White Cane Device is adequate for the proposed use, as reliable matching can be achieved even using images with quite a different perspective from the training image. Using the TSG-40-HTC dataset proved that the performance of White Cane Device is not affected when there is a change in viewpoint up to 30 degrees in either direction. Also, light occlusion contained within some testing and training images did not cause any observable drop in the system accuracy. White Cane Device received very good feedback from the intended end users of the system.

Chapter 6

FUTURE WORK

In this chapter we explore various ideas which could enhance the proposed system in some way, but, mainly due to the imposed time limitations, could not be added to our implementation. Other future work was not implemented as it was beyond the scope of this thesis.

6.1 IMPROVING THE CLIENT MOBILE APPLICATION

The client application can be made more efficient by adding the capability of sending the image to the server through 3G, as well as WIFI. This would enable the users to use White Cane App more flexibly, without requiring a WIFI connection. Also, the automatic setting of the image resolution would be more user-friendly. Another addition would be the addition of user preferences regarding to the format of the output result.

6.2 FEATURE EXTRACTION ALGORITHM

The implemented feature extraction algorithm can be optimized in many ways. For example, the use of color in the feature detectors and descriptors could result in better object recognition. Also, careful implementation of the algorithm could make the recognition process faster.

6.3 VIDEO OBJECT RECOGNITION

The use of video instead of images for object recognition, as proposed in [36], could help easier use by visually impaired persons as there is no need to focus on the object of interest. Rather, a number of snapshots are taken from the video as it is being recorded.

6.4 PATH INDICATION

A very interesting addition to the proposed system would be the implementation of algorithms which find the shortest path to the desired destination, given the current location. Visually impaired users would greatly benefit from step by step directions to a desired location.

6.5 CONCURRENT USERS

The full implementation of the White Cane Device prototype should be able to handle multiple concurrent users. Thus appropriate testing should be done to evaluate the scalability of such a system.

6.6 ADDITIONAL INFORMATION WITH TRAINING IMAGES

As suggested by Debono and Micallef in the client evaluation (Section 5.5), the addition of relevant information to training images would result in much more useful information to the end user. Such information could include the accessibility of a building and any other relevant information pertaining to the building in question. For example, a training image of a pharmacy might include information about its opening times.

6.7 Conclusion

The overall system could benefit from the mentioned additions in that it would become more efficient and user-friendly. Also, by providing more information, the system would give more useful guidance to the end users.

Chapter 7

CONCLUSIONS

Various state-of-the-art feature extraction algorithms were researched with the purpose of finding the best algorithm for this implementation. The evaluations of SIFT, SURF, PCA-SIFT and ASIFT algorithms were compared and SIFT resulted the algorithm best suited for the scope of this thesis. The features extracted by SIFT are scale, rotation and blur invariant, while being robust to changes in illumination and affine transformations. Such features were ideal for our implementation since we required catering for images which were taken by visually challenged persons. The geo-tagging capability of mobile phones was exploited to enable optimization of the object recognition process. The latter process involves extracting features from a query image and finding matches from a database of features extracted from training images.

Research was extended to papers proposing similar systems which implemented feature extracting algorithms as the ones mentioned above. Such papers provided a starting point to determine the order of the amalgamations of various processes required. The proposed system was modeled on a client-server environment, where the client is a mobile application enabling the user to take a picture and send it over to the server. The server, on the other hand, implements a webservice to act as an interface. This webservice accepts the picture taken by the user as well as the GPS coordinates. The server than performs object recognition on the picture and returns the results to the client.

White Cane Device resulted to be a very useful system capable of aiding visually impaired users by providing information about their current location. Extensive experiments and evaluations prove that White Cane Device is an effective solution which gives the correct matches for nearly all training images. Query images of a small resolution are correctly matched against a large database of features. The use of geo-tagged images enables the optimization of the object recognition process. Apart from considerably diminishing the time taken to compute the results, the GPS optimization also returns more correct results: less query images are matched incorrectly. Most importantly, a change in viewpoint did not result in any hindrance for the object recognition. Light occlusion was also not a concern.

The aims defined in Section 1.2 were fully achieved. White Cane Device is a very user-friendly system which can be used as a travelling aid to visually impaired persons. Apart from the latter, persons suffering from dementia could also find the proposed system useful. Also, White Cane Device can be tweaked for the use in the tourist industry; training images could be taken to be particular points of interest. For such a case, and also for the proposed scope of White Cane Device, additional information regarding the object depicted within the query images could be stored in the database. Thus any returned results would not only contain the name given to the training image, but also any other relevant information. For the proposed scope, this could

include the accessibility state of the place in question, while for the tourist industry this could include general information or historic context about the particular point of interest. White Cane Device is also flexible in the devices used as clients to connect to the server, as the client is only acting as a peripheral. Thus any camera enabled device with GPS capabilities could be used as a client, including tablet PC's.

This study proved to be a highly-academic exercise as well as an amazing tool to assist visually challenged persons. The application of image processing algorithms in combination with other location-based information provided an excellent basis to fruitfully train and employ an intelligent prototype system, thereby offering numerous future directions and additional research possibilities in such a noble and worthy area.

BIBLIOGRAPHY

1. Blindness software. *Nanopac.* [Online] [Cited: 01 12, 2011.] http://www.nanopac.com/Blindness%20Software.htm.

2. *Virtues of the Haversine.* **Sinnott, Roger W.** 1984, Sky Telesc., Vol. 68, p. 159.

3. **Yang, Ming-Hsuan.** Object Recognition. [ed.] Liu Ling and M. Tamer. *Encyclopedia of Database Systems.* s.l. : Springer US, 2009, pp. 1936-1939.

4. *Object recognition in the geometric era: A retrospective.* **Mundy, Joseph L.** s.l. : Springer, 2006. Toward CategoryLevel Object Recognition, volume 4170 of Lecture Notes in Computer Science. pp. 3-29.

5. **Roberts, Lawrence G.** *Machine Perception of Three-Dimensional Solids.* New York : Garland Publishing, 1963.

6. *Visual Perception by Computer.* **Binford, T. O.** 1971. Vol. ACM Annual Computer Science Conference.

7. **Agin, G. J.** *Representation and Description of Curved Objects.* Stanford Artificial Intelligence Lab Report. 1972.

8. *Symbolic Reasoning Among 3-D Models and 2-D Images.* **Brooks, Rodney A.** 1981, Artif. Intell., Vol. 17, pp. 285-348.

9. *Computing Exact Aspect Graphs of Curved Objects: Algebraic Surfaces.* **Ponce, Jean, Petitjean, Sylvain and Kriegman, David J.** 1992.

10. *Class-Based Grouping in Perspective Images.* **Zisserman, A., et al.** 1995. International Conference on Computer Vision.

11. *A network that learns to recognize three-dimensional objects.* **Poggio, T. and Edelman, S.** 1990, Nature, Vol. 343, pp. 263-266.

12. *Support vector machines for 3D object recognition.* **Pontil, M. and Verri, A.** 1998, Pattern Analysis and Machine Intelligence, IEEE Transactions on, Vol. 20, pp. 637-646.

13. *Selection of scale-invariant parts for object class recognition.* **Dorko, G., et al.** 2003, Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on, pp. 634-639.

14. *Object class recognition by unsupervised scale-invariant learning.* **Fergus, R., Perona, P. and Zisserman, A.** 2003. In CVPR. pp. 264-271.

15. *Indexing Based on Scale Invariant Interest Points.* **Mikolajczyk, Krystian and Schmid, Cordelia.** 2001. ICCV. pp. 525-531.

16. *Local Grayvalue Invariants for Image Retrieval.* **Schmid, Cordelia and Mohr, Roger.** 1997, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 19, pp. 530-535.

17. *Multi-view Matching for Unordered Image Sets.* **Schaffalitzky, Frederik and Zisserman, Andrew.** 2002. pp. 414-431.

18. *Distinctive Image Features from Scale-Invariant Keypoints.* **Lowe, David G.** 2004, International Journal of Computer Vision, Vol. 60, pp. 91-110.

19. *Simultaneous object recognition and segmentation by image exploration}.* **Ferrari, Vittorio, Tuytelaars, Tinne and Van Gool, Luc.** 2004. In Proceedings of the European Conference on Computer Vision.

20. *A Sparse Texture Representation Using Local Affine Regions.* **Lazebnik, Svetlana, Schmid, Cordelia and Ponce, Jean.** 2005, IEEE Trans. Pattern Anal. Mach. Intell., Vol. 27, pp. 1265-1278.

21. *Video Google: A Text Retrieval Approach to Object Matching in Videos.* **Sivic, Josef and Zisserman, Andrew.** 2003. ICCV. pp. 1470-1477.

22. **Se, Stephen, Lowe, David and Little, Jim.** Global Localization using Distinctive Visual Features. 2002.

23. **Brown, Matthew and Lowe, David G.** Recognising Panoramas. *ICCV.* 2003. pp. 1218-1227.

24. *PCA-SIFT: A More Distinctive Representation for Local Image Descriptors.* **Ke, Yan and Sukthankar, Rahul.** s.l. : IEEE Computer Society, 2004, Computer Vision and Pattern Recognition, IEEE Computer Society Conference on, pp. 506-513.

25. *ASIFT: An Algorithm for Fully Affine Invariant Comparison.* **Guoshen, Yu and Jean-Michel, Morel.** 2011, Image Processing On Line.

26. *SURF: Speeded Up Robust Features.* **Bay, Herbert, Tuytelaars, Tinne and Gool, Luc Van.** 2006. 9th European Conference on Computer Vision.

27. **Evans, Christopher.** *Notes on the OpenSURF Library.* University of Bristol. 2009.

28. *An Algorithm for Finding Best Matches in Logarithmic Expected Time.* **Friedman, Jerome H., Bentley, Jon Luis and Ari Finkel, Raphael.** September 1977, ACM Transactions on Mathematics Software, Vol. 3, pp. 209-226.

29. *Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces.* **Beis, Jeffrey S. and Lowe, David G.** 1997. In Proc. IEEE Conf. Comp. Vision Patt. Recog. pp. 1000-1006.

30. *Closest Point Search in High Dimensions.* **Nene, Sameer A. and Nayar, Shree K.** 1996. Proc. of IEEE Conference on Computer Vision and Pattern Recognition, (CVPR 96). pp. 859-865.

31. *Optimised KD-trees for fast image descriptor matching.* **Silpa-Anan, Chanop and Hartley, Richard.** 2008. 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 24-26 June 2008, Anchorage, Alaska, USA.

32. *Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration.* **Muja, Marius and Lowe, David G.** s.l. : INSTICC Press, 2009. International Conference on Computer Vision Theory and Application . pp. 331-340.

33. **Mikolajczyk, K.** [Online] [Cited: April 20, 2011.] http://www.robots.ox.ac.uk/~vgg/research/affine/.

34. *A Comparison of SIFT, PCA-SIFT and SURF.* **Juan, Luo and Gwon, Oubong.** 4, 2009, International Journal of Image Processing (IJIP), Vol. 3, pp. 143-152.

35. *A performance evaluation of local descriptors.* **Mikolajczyk, Krystian and Schmid, Cordelia.** 10, 2005, IEEE Transactions on Pattern Analysis & Machine Intelligence, Vol. 27, pp. 1615-1630.

36. *Mobile Visual Aid Tools for Users with Visual Impairments.* **Xu, Liu, Doermann, David and Li, Huiping.** [ed.] Xiaoyi Jiang, Matthew Y. Ma and Chang Wen Chen. s.l. : Springer, 2008. Mobile Multimedia Processing: Fundamentals, Methods, and Applications [outcome of the First International Workshop of Mobile Multimedia Processing (WMMP 2008), Tampa, Florida, USA, December 7, 2008]. Vol. 5960, pp. 21-36.

37. *Robust Real-Time Face Detection.* **Viola, Paul A. and Jones, Michael J.** 2004, International Journal of Computer Vision, Vol. 57, pp. 137-154.

38. *Mobile museum guide based on fast SIFT recognition.* **Ruf, Boris, Kokiopoulou, Effrosyni and Detyniecki, Marcin.** 2008. 6th International Workshop on Adaptive Multimedia Retrieval.

39. *Scene Recognition with Camera Phones for Tourist Information Access.* **Lim, Joo-Hwee, et al.** s.l. : IEEE, 2007. Proceedings of the 2007 IEEE International Conference on Multimedia and Expo, ICME 2007, July 2-5, 2007, Beijing, China. pp. 100-103.

40. *Geo-contextual priors for attentive urban object recognition.* **Amlacher, Katrin, et al.** 2009. pp. 1214-1219.

41. *Human and machine recognition of faces: a survey.* **Chellappa, R., Wilson, C. L. and Sirohey, S.** May 1995, Proceedings of the IEEE, Vol. 83, pp. 705-741.

42. [Online] [Cited: May 25, 2011.] http://pixhawk.ethz.ch/wiki/software/computer_vision/surf_features.

43. *Parallelization of video processing: From programming models to applications.* **Lin, Dennis, et al.** 6, 2009, IEEE Signal Processing Magazine, Vol. 26, pp. 103-112.

44. *A Parallel Hardware Architecture for Scale and Rotation Invariant Feature Detection.* **Bonato, V., Marques, E. and Constantinides, G. A.** 2008, Circuits and Systems for Video Technology, IEEE Transactions on, Vol. 18, pp. 1703-1712.

45. *Learning a Sparse Representation for Object Detection.* **Agarwal, Shivani and Roth, Dan.** s.l. : Springer, 2002. Computer Vision - ECCV 2002, 7th European Conference on Computer Vision, Copenhagen, Denmark, May 28-31, 2002, Proceedings, Part IV. Vol. 2353, pp. 113-130.

46. [Online] [Cited: May 19, 2011.] http://dib.joanneum.at/cape/TSG-40/index.php?page=samples.

47. [Online] [Cited: May 19, 2011.] http://dib.joanneum.at/cape/TSG-40/index.php?page=tech.

48. [Online] [Cited: April 18, 2011.] http://haft2.com/haft2know/2008/04/mothers_day.html.

49. [Online] [Cited: April 15, 2011.] http://www.scielo.br/scielo.php?pid=S0104-65002009000300002&script=sci_arttext.

50. [Online] [Cited: May 2, 2011.] http://www.blogcdn.com/mobile.engadget.com/media/2010/05/htc-desire-orange-black.jpg.