LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY

COARSE GLOBAL VISUAL LOCALIZATION OF A MOBILE ROBOT

Examiners: Docent Ville Kyrki, M.Sc. Olli Alkkiomäki
Supervisor: Dr. Tech. Ville Kyrki

Lappeenranta, May 25th, 2007

Almasi S. Maguya
Ruskonlahdenkatu 13-15 C11
53850 Lappeenranta
Finland

Tel. +358 408 159489

almasi.maguya@lut.fi

ABSTRACT

Lappeenranta University of Technology
Department of Information Technology

Almasi Maguya

Coarse Global Visual Localization of a Mobile Robot

Master's thesis

2007

85 pages, 28 figures, 9 tables and 2 appendices.

Examiners: Docent Ville Kyrki,
M.Sc. Olli Alkkiomäki.

Keywords: mobile robot localization, coarse localization,global localization,visual localization, interest point detectors, local features, image matching, interest point descriptors.

Localization, which is the ability of a mobile robot to estimate its position within its environment, is a key capability for autonomous operation of any mobile robot. This thesis presents a system for indoor coarse and global localization of a mobile robot based on visual information.

The system is based on image matching and uses SIFT features as natural landmarks. Features extracted from training images are stored in a database for use in localization later. During localization an image of the scene is captured using the on-board camera of the robot, features are extracted from the image and the best match is searched from the database. Feature matching is done using the k-d tree algorithm.

Experimental results showed that localization accuracy increases with the number of training features used in the training database, while, on the other hand, increasing number of features tended to have a negative impact on the computational time. For some parts of the environment the error rate was relatively high due to a strong correlation of features taken from those places across the environment.

PREFACE

This masters thesis work was carried out in the department of Information Technology at Lappeenranta University of Technology as a partial fulfilment of the requirements for the degree of Master of Science in Technology.

Although writing a thesis is an indication that one is at the final stages of his or her studies, many will agree with me it the most difficult and daunting stage of studies. This is because it is filled with more questions than answers–where do I start? How do I go about? Will I finish in time? not to mention the failures and frustrations–code not compiling,mental blocks and poor results which are rife during this period.

Despite all these difficulties I was able to finish my work on time. This would not have been easy without the support and encouragement I have received throughout my studies, which ought to acknowledge.

I would like to take this chance to thank my supervisor Dr. Tech. Ville Kyrky for guiding me throughout the course of my work on both technical and non-technical matters.

Thanks, too, to the coordinators of the IMPIT program and Lappeenranta University of Technology in general for the financial support I have received during my studies.

Finally, many thanks go to my family for the encouragement and support you have shown me for the whole period I have been away from home.

Lappeenranta, May 25th, 2007

Almasi S. Maguya

TABLE OF CONTENTS

ABBREVIATIONS AND SYMBOLS

2D	Two dimensional
3D	Three dimensional
AFV-I	Autonomous Flying Vehicle-I
BBF	Best-Bin-First
CBIR	Content Based Image Retrieval
CGI	Common Gateway Interface
det	Determinant
DoG	Difference of Gaussian
DOGSS	Difference Of Gaussian Scale Space
EKF	Extended Kalman Filter
FTP	File Transfer Protocol
FTPS	Secure file transfer protocols
GDK	GTK+ Drawing Kit
GSL	Gnu Scientific Library
GSS	Gaussian Scale Space
GTK	GIMP Toolkit
HTTP	Hypertext Transfer Protocol
I	Image
ID	Identity
LDAP	Lightweight Directory Access Protocol
LVSM	Location Vector Space Model
MLC	Monte Carlo Localization
MMW	Millimetre-wave
MEX	Matlab executables
pgm	portable graymap (file format)
RANSAC	Random Sample Consensus
ROC	Receiver Operating Characteristics
SDS	Scientific Data Systems
SIFT	Scale Invariant Feature Transform
SLAM	Simultaneous Localization and Mapping
SSD	Sum of Squared Differences
SURF	Speeded Up Robust Features

K	Kilo
$	Shell prompt
>>	Matlab prompt
λ	Eigenvalue

1 Introduction

1.1 Background

A mobile robot is a free-roving robot that is able to move through space to accomplish some task [21]. The first mobile robots appeared during World War II [34]. These were mostly in form of flying bombs which had the capability to sense their proximity to their targets.

From 1966, the Artificial Intelligence Center at Stanford Research Institute began research on a project named "Application of Intelligent Automata To Reconnaissance" [39, 50]. The aim of the project was to develop a mobile robotic system nicknamed "Shakey", that would have limited capability to perceive and model its environment based on artificial intelligence techniques. In 1969, demonstrations of Shakey were conducted and in 1970 Shakey was introduced as the first mobile robot controlled by artificial intelligence – making a breakthrough in the field of autonomous mobile robots. Shakey was controlled by SDS-940 computer acquired in 1966 with 64K 24-bit words of memory and was capable of performing tasks that required planning, route finding and rearranging of simple objects[50].

Unlike traditional industrial robotic systems which are made up of linked manipulators and are fixed to one physical location, mobile robots have the capability to move around in their environment and are not fixed to one physical location. Figure 1 shows pictures of a mobile robot and a robotic arm.

It is because of the fact that mobile robots are capable of interacting with their environment autonomously that today we see mobile robots being used to perform different kinds of tasks. These include both tasks in structured indoor environments and unstructured outdoor environments. With the former, applications include for example the transportation industry— Automatic Guided Vehicles[44], cleaning – large buildings and household[44][19, p.109–111]; with the latter, applications include agriculture – combine harvesters[7, p.572], space exploration and performing tasks which are dangerous to human beings for example handling of toxic materials [42] and military applications—demining and surveillance[27].

It should now have become evident that there are many potential applications for

(a) Pioneer 3-DX mobile robot (b) Lynx 5 robotic arm

Figure 1: Examples of a mobile robot and a robotic arm

mobile robots. For example, In [20] it is estimated that in countries like Sweden the number of retired people will increase by over 50% in the next 30 years. These elderly people will need assistance in their day to day activities such as dressing and preparing meals. Because of this potential there has been a great deal of research attention to this area in the past decade.

In order for a mobile robot to perform a task successfully it needs to autonomously move around its environment. This capability is referred to as navigation.

Navigation is one of the most challenging skills required from a mobile robot [42]. Successful navigation depends on four factors:

1. *Perception* – Ability of the robot to interpret its sensor input to extract meaningful data.
2. *Cognition* – Ability of the robot to decide how to act to achieve its goals.
3. *Motion control*– Ability of the robot to modulate its motor output to achieve desired trajectory.

4. *Localization* – Ability of the robot to determine its position within its environment.

Of all the four factors, localization has received the greatest research attention in the past decade [42]. This can be attributed to the challenging nature of the localization problem. Localization is based on sensor data but sensors suffer from errors; for example a camera taking an image of a scene will give different result under different illumination conditions. Another problem with sensors is aliasing—non uniqueness of sensor readings taken from the same environmental state [42].

1.2 Objectives and scope

The goal of this thesis was to develop a visual-based localization system for a mobile robot. The system is limited to non-metric localization, the aim was to give the robot the capability to coarsely and globally localize itself in an indoor environment. Localization will be coarse in the sense that the robot only needs to be able to recognize a place it has seen before in the environment where it operates. These places are referred to as nodes. Thus a node can be, for example, a point in a room or a corridor at which the robot should be able to recognize the place. On the other hand, localization will be global in the sense that the robot should be able to localize itself without the need of knowledge of its previous position. Figure 2 shows the floor plan of the environment in which the robot is going to be operating.

The system is based on image matching whereby SIFT features are used as natural landmarks . SIFT features are extracted from training images and stored in a feature database. During localization an image of an unknown scene is captured using the on-board camera of the robot and SIFT features are extracted from the image. These features are used to find the closest match to the image by matching them against all the features stored in the feature database and finding the image which gives the highest number of matches.

The robot which is going to be used in this project is shown in figure 3. It is a Pioneer 3-DX mobile robot system located in the laboratory of machine vision at Lappeenranta University of Technology.

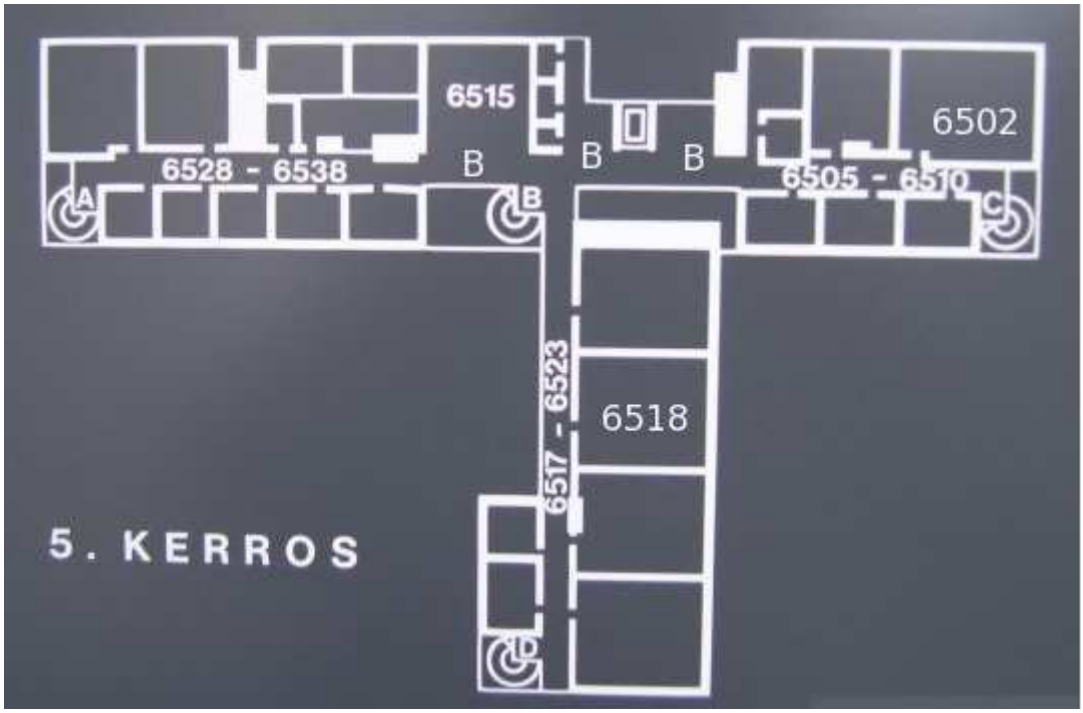


Figure 2: Floor plan of the robot environment. The labels are as follows:

- 6518 : Masters students' office (N1)
- 6517 - 6523 : Corridor I (N2)
- B : Stair case and main corridor (N3)
- 6502 : Machine vision laboratory (N4)
- 6505 - 6510 : Corridor II (N5)
- 6515 : Meeting room (N6)
- 6528 - 6538 : Corridor III (N7)

Figure 3: The test robot.

The robot is equipped with an on-board video camera which has zoom and pan-tilt capabilities. The robot is also connected to a computer running the Linux operating system.

1.3 Structure of the thesis

The thesis is structured as follows: Section 2 gives a survey of various localization techniques presented in the literature and their pros and cons. In section 3 interest points and interest point detectors are surveyed and examples are presented of some open implementations of the algorithms of the latter. After exploring interest points and interest point detectors in section 3, a general overview of the localization system is given in section 4 whereby the constituent modules of the system are described together with the software tools used in implementing the system. Sections 5 and 6 are reserved for experiments and results while sections 7 gives the conclusions, final thoughts and recommendations for future work.

2 Mobile robot localization

Mobile robot localization, which is also known as position estimation, is the process of determining the pose of a robot relative to a given map of the environment. Localization is the most basic perceptual problem in robotics– nearly all robotic tasks require knowledge of the location of objects that are being manipulated[46, p.191].

The level of difficulty of a localization problem depends on several factors. These include, for example, the nature of the environment the robot is operating, availability of initial information relative to the localization problem, the number of robots involved in the localization process and whether or not the localization system controls the motion of the robot[46, p.193-196]. In the following sections a taxonomy of localization problems based on these factors will be discussed.

2.1 Passive and active localization

A localization problem can be characterized as passive or active depending on whether or not the localization algorithm controls the motion of the robot [46, p.195]. In passive localization the localization algorithm does not control the motion of the robot, instead the robot is controlled by some other means and the localization algorithm only observes the robot operating, that is, the robot motion is not aimed at facilitating the localization process.

In active localization the localization algorithm has direct control over the robot motion. In this case the robot motion is aimed at facilitating the localization process by trying to minimize the localization error.

Generally, in passive localization problems the robot is given a map of the environment , such as the one shown in figure 4(b), and a history information state, η_k, and computes the non-deterministic information state $X_k(\eta_k)$ in X. In active localization problems some k and a sequence of actions $u_1, ..., u_{k-1}$ are computed such that the non-deterministic information state is as small as possible.

In general, active localization approaches give better localization results because of their direct control over the motion of the robot. Using this capability, the localization

module can move the robot out of a difficult situation in which localization is very difficult, to a place where the robot can localize itself easily.

An example of a situation in which active localization is superior to passive localization is in a problem which involves symmetries [23, p.659]. In this case a map similar to figure 4(b) is given to the robot for localization purposes.

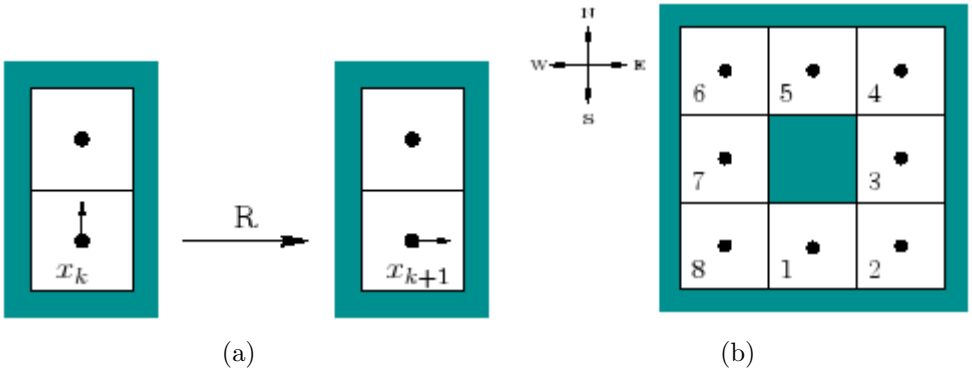


Figure 4: (a)The robot tries to move in the north direction where there is an obstacle hence the robot changes its direction but not its position. (b)A map given to the robot for localization[23].

The robot in question may be oriented in one of the four directions: N, E, W and S, which stand for "north," "east," "west" and "south" respectively. Also, although the robot is treated as a point, its orientation is important because it does not have a compass–if it chooses to move in a particular direction it does not necessarily know which direction it will be heading with respect to the four directions.

Furthermore, the robot is given four actions,

$$U = F, B, R, L \tag{1}$$

which stand for "forward," "backward," "right motion" and "left motion" respectively. These motions occur with respect to the current orientation of the robot (which may

be unknown) as follows:

1. *F action* – The robot moves forward and maintains its orientation.
2. *B action* – The robot changes its orientation by 180° and then moves forward one grid element.
3. *R action* – The robot turns right by 90° and then moves forward one grid element.
4. *L action* – The robot turns left 90° and then moves forward one grid element.

The robot is equipped with a simple sensor that can detect whether the robot moved in a direction attempted successfully. If it is not possible to move in a particular direction because of an obstacle the robot changes its orientation (in the case of B, R or L) but does not change its position, see figure 4(a).

Now suppose that the robot is initially in position 1 (see figure 4(b)) facing east and the action sequence (F,L,F,L,..) is executed. It can be seen that the robot will travel around in cycles. The same behaviour is also observed when the action sequence (F,L,F,L,..) is executed while the robot is in the following positions :

- Position 3 facing north.
- Position 5 heading west.
- Position 7 heading south.

It can be seen the state space in this example contains 32 states (four directions at each position). However, after executing some motions the non-deterministic state can be reduced to a symmetric class of no more than 4 possible states. Thefore, in the best case, $X_k(\eta_k)$ might become a singleton set, which means that the robot knows its position and orientation on the map. However, due to symmetries, as it has been illustrated above, it might not be possible for the robot to determine its pose[23, p.642].

2.2 Local and global localization

Depending on the type of information that is available initially and at runtime, a localization problem can be characterized as local or global.

Local localization, also known as position tracking, assumes that the initial position of the robot is known and tries to track the robot's pose relative to this position[6][46, p.193]. In this case localization of the robot can be achieved by accommodating the noise in robot motion. Since the effect of such noise is usually small, methods for position tracking often rely on the assumption that the pose error is small and approximate the pose uncertainty by a unimodal distribution, such as a Gaussian[46, p.194].

As an example, in [6] position probability grids are used for position tracking. For a given initial position the method keeps track of the robot's current position by matching sensor readings against a metric model of the environment. The method is designed to work with noisy sensors and approximate models of the environment and is capable of integrating sensor readings of different types of sensors over time. In contrast to other position tracking approaches, the method is not restricted to a fixed set of pre-defined features such as doors, because it can extract arbitrary features of the environment using raw sensor data.

In global localization the initial pose of the robot is unknown, instead the robot is placed somewhere in its environment without any prior position estimate[46, p.194]. Global localization problems are more difficult than position tracking as no assumptions are made about the boundedness of the pose error. A family of probabilistic algorithms known as Monte Carlo Localization (MLC) is among the most popular methods for solving the global localization problem[30].

In [30] an approach is proposed that extends the MCL algorithm by addressing its problems when localizing in highly symmetrical environments. The approach proposed introduces the idea of clusters of particles and modifies the proposed distribution to take into account the probability of a cluster of similar poses. Each cluster is considered to be a hypothesis of where the robot might be located and is independently developed using the MCL algorithm–effectively leading to a particle filter that works at two levels, that is, the particle level and the cluster level.

Sometimes a robot may get kidnapped and teleported to some other location during its

operation. In this situation the robot may think it knows where it is while it does not. This phenomenon is known as the kidnapped robot problem[46, p.194]. The kidnapped robot problem is a variant of the global localization problem but it is more challenging as the robot gets confused. In global localization, on the other hand, the robot knows that it does not know where it is.

In addition to the factors discussed above are two more factors which also influence the degree of difficulty a localization problem posses. These include the nature of the environment in which the robot is operating and the number of robots involved in the localization process.

Environments can be static or dynamic. Static environments are those in which the only variable quantity is the robots pose, that is, only the robot moves in the environment while dynamic environments are those in which objects other than the robot are moving. Examples of these objects include people, movable furniture and daylight (for robots equipped with a camera)[46, p.194]. In general, localization in dynamic environments is more difficult than localization in static environments.

The number of robots involved in localization also has an impact on the hardness of the problem. Single robot localization is the most commonly studied approach to localization and it deals with a single robot only. Multi-robot localization, on the other hand, deals with teams of robots[46, p.196]. Multi-robot localization is more challenging because it involves communication between the robots involved.

Due to the fact that not all localization problems are equally challenging there is no single technique which can work on all kinds of problems. Each problems poses its own challenges and hence needs methods specifically designed to address these challenges. In the following section some of the methods proposed in the literature will be reviewed.

2.3 Localization techniques

Several localization techniques have been proposed in the literature. In general these methods fall in the following three main categories[14]:

1. Behaviour-based approaches.

2. Landmark-based approaches.

3. Dense sensor matching approaches.

These methods are discussed below.

2.3.1 Behaviour-based approaches

This category of localization methods relies on the interaction of the robot actions with the environment to navigate[14]. For example, [31] used a behavioural control architecture for controlling the AFV-I, a flying robot that was capable of locating and manipulating objects and transport them from one location to another under hazardous conditions, without human guidance and within a fixed time limit.

Behaviour based approaches solve the localization problem in a parallel fashion–each behaviour, acting concurrently with other behaviours, extracts from the environment only the information required to complete a given task at a given time[31]. This approach does not need a global world model of the environment, thus has the advantage of reducing the robot's computational load.

One drawback of these approaches is their limited ability to localize the robot geometrically. This is because their navigation capability is implicitly in their sensor/action history[31].

2.3.2 Landmark based methods

Landmarks are passive objects in the environment that provide a high degree of localization accuracy when they can be viewed by the robot[42, p.245]. Landmark-based methods rely on the recognition of landmarks to keep the robot localized geometrically[14]. Landmarks may be provided beforehand or the robot may have to learn them in the process of mapping the environment.

Landmark-based localization is in two forms. The first one uses artificial landmarks and the second one uses natural landmarks. Artificial landmarks are used in known environments where as natural landmarks are used in unknown environments.

Artificial landmark-based localization methods are effective and easy to implement. This is because landmarks can be selected in such a way that they can be easily detected in the images. One drawback of these methods is that they require introduction of landmarks in the environment, something which may not be desirable in home or office environments. Additionally, detection of landmarks can be affected by such common image transforms as scale, causing errors in localization.

Examples of artificial landmark-based localization are those proposed in [24, 1]. In [24] ID tags were used as artificial landmarks for localization of an indoor mobile robot. Unique ID tags were placed at distinct locations in the environment and a web camera was used to detect the nearest tag, which was used to estimate the position of the robot. Similarly, in [1] bar codes were used as artificial landmarks and the position of the robot was estimated by reading the bar codes with a camera.

In natural landmark-based localization a robot localizes itself by observing and extracting relevant information about the elements in the environment [8]. Examples of such elements include walls, furniture, doorways and windows.

A good example of work based on natural landmark localization is that presented in [18]. Landmarks used were planar quadrangular surfaces. The landmarks were detected by edge detection using classical contour extraction and segmentation methods. Segment couples corresponding to potential landmarks are formed according to geometric and luminance consistency criteria and matching is performed using partial Hausdorff distance.

There is also a considerable body of research work which uses interest points as natural landmarks for localization of mobile robots. Interest points are robust, invariant to common image transformations such as scale, illumination and changes in viewpoint. Additionally, there exist algorithms which can compute interest points in a near real-time speed, for example [26]. All these characteristics make interest points particularly interesting for visual localization of mobile robots.

Another interesting example which uses interest points as natural landmarks is [5]. In this example a method is proposed for metric localization using Scale-Invariant visual features using a single perspective camera. 2D maps of scale-invariant features are used as visual landmarks. To build the maps the Scale Invariant Feature Transform (SIFT) is used to detect features in images. A robot equipped with a perspective

camera and a laser range finder is steered through the environment to obtain the image data as well as proximity and odometry measurements. Because the number of SIFT features extracted after this step is very large, the features are down-sampled to obtain a reduced set of features that is used for localization. During localization, close-by particles are clustered and for each cluster the set of potentially visible features in the map is estimated using ray-casting. These relevant map features are then compared to the features extracted from the current image.

Similarly, in [49] Wang et al. propose a coarse-to-fine vision based localization scheme using Harris-Laplace interest points characterized by Scale Invariant Feature Transform (SIFT) descriptors as natural landmarks. The method operates in two stages, in the first stage (Coarse localization) the rough location of the robot is estimated using information indexed in a location vector space model database (LVSM). This stage is fast but not accurate. In the second stage (Fine localization) a voting algorithm is used against a location database to give a more accurate position of the robot. The later stage is relatively slow but more accurate.

2.3.3 Dense sensor matching approaches

This category of localization techniques relies on sensor information to update the pose of the robot. This is done by matching dense sensor scans against a surface map of the environment[14]. Since no landmarks are extracted this approach can take advantage of whatever features are present in the environment.

In these methods localization involves determining the probability of the robot being at pose l given sensor inputs s_n and a history $a_1, ..., a_n = A^n$ of executed actions[14, 11].

Markov localization and Kalman filter localization are examples of localization methods that fall under this category. Markov localization uses explicitly specified probability distribution across all possible robot positions while scan matching uses a simple Gaussian for the distribution. Kalman filter localization, on the other hand, uses a Gaussian probability density representation of the robot and scan matching for localization [42, p.214,227][14].

Both Markov localization and Kalman filter localization are very popular in mobile robot research for indoor environment navigation.

Gutman [15], combined the Markov and Kalman localization methods to take advantage of experimental results which showed that in general grid-based Markov localization is more robust than Kalman filter localization while Kalman filter localization can be more accurate than the former. This method is suitable for robots observing known landmarks and is reported to outperform both of its underlying methods. The method, however, suffers from computational complexity.

2.4 Map building

The localization techniques discussed above need a map of the environment in which the robot operates for localization. Thus there is a need for a method to build maps for localization purposes.

In order to autonomously create a map of an environment the robot must know where it is within the environment(localize itself) and in order know where it is the robot needs a map of the environment. In order to achieve this the robot must simultaneously build a map of the environment and use the same map it is building to localize itself. This problem is known as Simultaneously Localization and Mapping(SLAM)[9].

In SLAM a robot needs to be able to navigate without prior knowledge of the environment, that is, a robot starts without any map and without any idea of its whereabouts and starts moving while simultaneously creating a map of the environment and use the same dynamically changing map to localize itself. This eliminates the need for both artificial infrastructures and and a prior topological knowledge of the environment[9]. A solution to the SLAM problem is therefore of paramount importance to applications where absolute position or precise map information can not be obtained. These include, for example, autonomous planetary exploration, subsea autonomous vehicles, autonomous air-borne vehicles, and autonomous all-terrain vehicles in tasks such as mining and construction[9].

A range of techniques have been proposed for addressing the SLAM problem, the most popular being the Kalman filter approach. One of the reasons why this approach is popular is because it provides a recursive solution to the navigation problem[9].

An an example of an extended Kalman filter (EKF) based SLAM algorithm is presented

in [9]. The algorithm was implemented on a standard vehicle operating in an outdoor environment and equipped with millimeter-wave (MMW) radar to provide relative map observations.Similarly in [45] straight lines are used in an extended Kalman filter SLAM system in a manner that integrates easily with point features. In order to achieve real-time operation a fast line detection algorithm is employed. The line detection algorithm does not detect all the lines in a frame but enough to initialize the system.

In [41] Stephen et al. propose a method for map building using SIFT features. SIFT features are kept in a database and are later tracked while building a 3D map of the environment and using the same map for localization. An interest thing about this approach is that the map is built by merging and aligning several 3D submaps. The map built is also used for global localization. This is achieved by matching groups of descriptors to the map using the Hough transform and RANSAC.

3 Interest points

The system proposed in this thesis employs interest points for localization, therefore a review of interest points is given in the following sections.

Interest points are locations in an image where the signal changes two-dimensionally [40]. Examples include corners, blobs and T-junctions. According to [38], an interest point is described as a point in an image having the following general characteristics:

1. It has a clear, preferably mathematically well formed, definition.
2. It has a well defined position in an image.
3. The local image structure around an interest point is rich in terms of local information contents, such that the use of interest points simplify further processing in the vision system.
4. It is stable under local and global perturbations in the image domain, including deformations such as those arising from perspective transformations (sometimes reduced to affine transformations, scale changes, rotations and/or translations) as well as illumination/brightness variations, such that the interest points can be reliably computed with high degree of reproducibility.
5. Optionally, the notion of interest point should include an attribute of scale, to make it possible to compute interest points from real-life images as well as under scale changes.

Interest points provide robust and stable image features which have a wide range of applications in machine vision. Examples of applications include image retrieval [28], mobile robot localization [49] and object recognition [25]. Others include 3D reconstruction, motion tracking and robot navigation [12].

While interest points detectors are used to detect points of interest in an image, feature descriptors are used to represent the features in a highly distinctive and invariant way.

Many different methods for interest point detection and feature description have been proposed in literature, some of these will be discussed in the following sections.

3.1 The Harris corner detector

The Harris corner detector is introduced here as an old but still very widely used interest point detector. The Harris corner detection algorithm [17] is a result of the work of Chris Harris and Mike Stephens whose goal was to use computer vision to understand the unconstrained 3D world. The image features required for the task needed be discrete , for this reason their work concentrated on extraction and tracking of feature-points or corners.

The algorithm is based on the Moravec corner detector [32]. Instead of using a binary window function, which leads to noisy response, the algorithm uses a Gaussian function

$$w(x,y) = exp\left(-\frac{x^2+y^2}{2\sigma^2}\right). \tag{2}$$

For a two dimensional grey-level image I, if we take a small patch around (u,v) and shift it by (x,y), then the SSD between the two patches, S, is given by

$$S = \Sigma_u \Sigma_v (I(u,v) - I(u-x, v-y))^2. \tag{3}$$

For a point p at (x,y) the Hessian of S gives a matrix C defined as

$$C = \begin{bmatrix} \Sigma I_x^2 & \Sigma I_x I_y \\ \Sigma I_x I_y & \Sigma I_y^2 \end{bmatrix} \tag{4}$$

where the sums are taken in the neighbourhood of p [47].

The matrix C is symmetric and hence can be diagonalized by the rotation of the coordinate axes, thus C can be thought of as a diagonal matrix

$$C = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \tag{5}$$

where the two eigenvalues λ_1 and λ_2 are both nonnegative.

The presence of a corner at a point p can be determined by considering the geometric interpretation of λ_1 and λ_2 as follows:

1. $\lambda_1 = \lambda_2 = 0$: The area around p is uniform hence the image gradient vanishes everywhere making C a null matrix – no feature of interest at this point.
2. $\lambda_2 = 0, \lambda_1 > 0$: This denotes the presence of an edge.
3. $\lambda_1 \geq \lambda_2 > 0$: There is a corner at p.

Since the computation of λ_1 and λ_2 is computationally expensive, Harris and Stephens proposed the use of a corner response function, $\mathbf{R}$, given by

$$\mathbf{R} = det(C) - k(trace(C))^2 \tag{6}$$

where $detC = \lambda_1\lambda_2$,
trace $C = \lambda_1 + \lambda_2$ and
k is an empirical constant.

The Harris corner detection algorithm is summarized in 1.

Figure 5 shows an example of an image and corners detected on it using the Harris corner detector.

The Harris corner detector is invariant to image rotation since rotation does not affect the eigenvalues λ_1 and λ_2, but non-invariant to changes in image scale.

3.2 Scale Invariant Feature Transform (SIFT)

Scale Invariant Feature Transform (SIFT) is a method for detecting, extracting and representing of local image features or interest points [26] which makes it both a feature detector and descriptor. The features detected by SIFT are known as SIFT features

Algorithm 1 The Harris corner detector

1. Compute the x and y derivatives of an image
   $I_x = G^x_\sigma * I \qquad I_y = G^y_\sigma * I$
2. Compute the products of derivatives at every point
   $I_{x2} = I_x I_x \qquad I_{y2} = I_y I_y \qquad I_{xy} = I_x I_y$
3. the sums of the products of derivatives at each pixel for a square window neighbourhood.
   $S_{x2} = \Sigma I^2_x \qquad S_{y2} = \Sigma I^2_y \qquad S_{yx} = \Sigma I_x I_y$
4. At each pixel (x,y) form the matrix C given by
   $C = \begin{bmatrix} S_{x^2} & S_{xy} \\ S_{xy} & S_{y^2} \end{bmatrix}$
5. Compute the response of the detector at each pixel
   $\mathbf{R} = det(C) - k(trace(C))^2$
6. Threshold on value of **R**

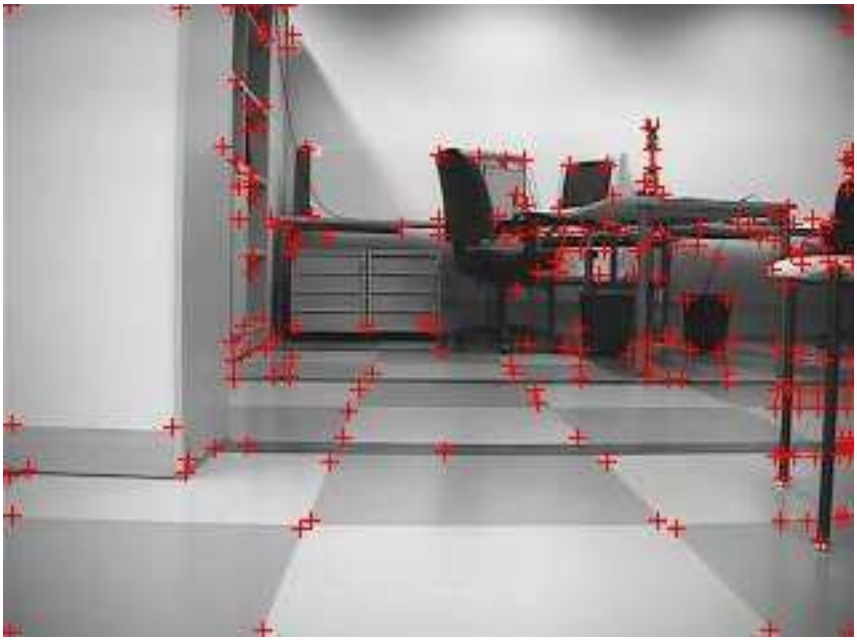


Figure 5: An example of corners detected by the Harris corner detector

and are characterized by invariance to some common image transforms including image scaling and rotation. The features are also invariant to changes in illumination and 3D camera viewpoint.

Because SIFT features are well localized in the spacial and frequency domains, the probability of disruption due to noise, occlusion and clutter is very low [26].

SIFT features are highly distinctive, which means a single feature can be matched with a high probability against a large database of features. This characteristic makes SIFT features suitable for such applications as object recognition [25], mobile robot localization [49] and image retrieval, which are all based on image matching.

To minimize the cost of extracting the features, a cascade filtering approach is employed whereby more expensive operations are applied only to areas which pass an initial test.

3.2.1 Detection and extraction of SIFT features

The process of detecting and extracting SIFT features can be broken down to four main stages:

1. Scale-space extrema detection
2. Keypoint localization
3. Orientation assignment
4. Generation of keypoint descriptors

In the following sections these stages will be discussed in more detail.

3.2.2 Scale-space extrema detection

This stage aims at identifying locations and scales that are invariant to scale changes. This is achieved by searching for candidate locations across all possible scales using a continuous function of scale known as scale space.

SIFT uses the difference-of-Gaussian (DoG) function for scale space because it is both efficient and stable. DoG is also a close approximation to the scale-normalized Laplacian of Gaussian, $\sigma^2 \bigtriangledown^2 G$.

The scale space of an image $I(x, y)$ is defined as a function $L(x, y, \sigma)$ that is obtained by convolving the image with a variable-scale Gaussian, $G(x, y, \sigma)$ [26]

$$L(x, y, \sigma) = G * I(x, y) \tag{7}$$

where $*$ represents the convolution operator in x and y and

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \tag{8}$$

In order to locate stable keypoint locations in scale space, the image is convolved with scale-space extrema in the difference-of-Gaussian function

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \tag{9}$$

which gives

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma) \tag{10}$$

This function is efficient to compute because it can be computed by image subtraction of smoothed images L, which need to be computed in any case for scale-space feature description.

Figure 6 illustrates the approach used to compute the difference-of-Gaussian images , $D(x, y, \sigma)$. Processing is done in stages known as octaves. In each octave the initial image is incrementally convolved with Gaussians to produce images separated by a constant factor k in scale-space. Then adjacent image scales are subtracted to give the difference-of-Gaussian images.

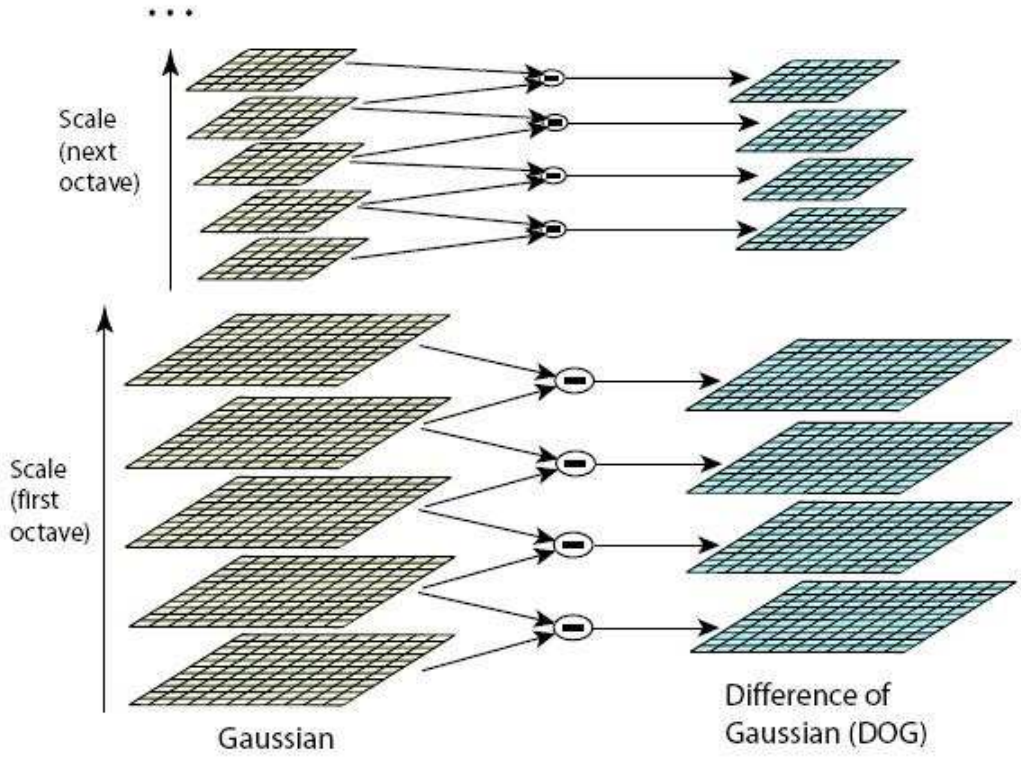


Figure 6: Computation of difference-of-Gaussian images in SIFT[26].

In the next step, the Gaussian blurred image that has twice the initial value of σ is re-sampled by taking every second pixel in each row and column as shown in figure 7.

3.2.3 Local extrema detection

Interest points are identified as local maxima or minima of the difference-of-Gaussian function, $D(x, y\sigma)$.

Detecting the local maxima and minima is done by comparing each sample point to its eight neighbours in the current image and nine neighbours in the scale above and below, this is shown in figure A point is selected only if it is either larger than or smaller than all of its neighbours. 7.

This stage requires determination of suitable sampling frequency in both the image and

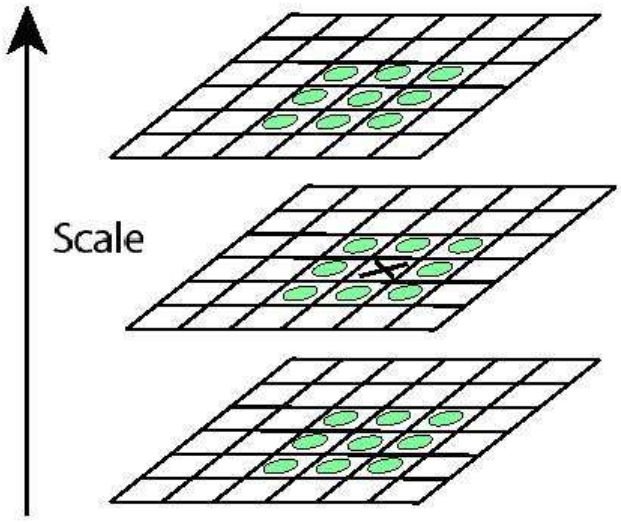


Figure 7: Local extrema detection in SIFT[26].

scale domains. Determination of suitable sampling frequency is essential for reliable detection of the extrema of the difference-of-Gaussian function, $D(x, y, \sigma)$.

Since there is no minimum spacing of samples that can detect all extrema, the sampling frequencies were determined experimentally [26]. Experiments performed in [26] for determining the sampling frequencies used a collection of images taken from a wide range of environments, for example industrial images, aerial photographs and outdoor scenes.

To determine the frequency of sampling in the scale space each of the test images is subjected to a range of image transformations including rotation, scaling, affine stretch, brightness and contrast changes and addition of image noise.

The experiments performed showed that the scale space difference-of-Gaussian function has a large number of extrema and it is very expensive to detect all of them but the most stable and useful subset of the extrema can be detected with a coarse sampling of scales [26].

Frequency of sampling in the spacial domain involved determination of the required

amount of pre-smoothing, σ , that is to be applied to each image level before building the scale space representation for an octave. Although results showed repeatability to be increasing with σ, using large σ is costly and in [26] the value of σ used was 1.6 which provided close to optimal repeatability.

3.2.4 Keypoint localization

The goal of this stage is to eliminate points that have low contrast or are poorly localized along the edges.

To eliminate extrema with low contrast the Laplacian value for each keypoint which was found in the first stage is computed. The location of the extrema, $\mathbf{z}$, is given by

$$\mathbf{z} = -\frac{\partial^2 D^{-1}}{\partial x^2}\frac{\partial D}{\partial x} \tag{11}$$

All the extrema at which the value of $\mathbf{z}$ is below a given threshold [1] are discarded.

Extrema with poor localization are detected and eliminated based on the fact that in these cases the difference-of-Gaussian function will have a large principal curvature across the edge but a small one in the perpendicular direction. The principal curvatures are computed from a 2 x 2 Hessian matrix

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{12}$$

Keypoints which have a ratio between the principal curvatures greater than a given threshold are discarded.

[1] The threshold is to be determined experimentally, for example in [26] all keypoints with value of $\boldsymbol{z}$ below 0.03 were eliminated.

3.2.5 Orientation assignment

In this stage a consistent orientation is assigned to each keypoint based on local image properties. The keypoint descriptor can then be represented relative to this orientation, achieving rotational invariance.

Based on experiments, it was found in [26] that the approach represented in 2 is more appropriate with respect to the stability of the results.

Algorithm 2 Assigning orientations to keypoints.

---

1. Select the Gaussian smoothed image, $\boldsymbol{L}$, which has the closest scale using the scale of the keypoints in order for all computations to performed in a scale-invariant manner.

2. Compute the gradient magnitude, m, and orientation, θ for each sample L(x,y) at this scale

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta = tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y)))$$

3. Form an orientation histogram from the gradient orientation of the sample points within the region around the keypoint.

4. Locate the highest peak in the histogram. Use this peak together with any other peak that is within 80% of it to create a keypoint with that orientation.

5. Fit a parabola to the 3 histogram values closest to each peak to interpolate the peak's position.

3.3 The SIFT keypoint descriptor

The local gradient data computed in the previous stage is used to create keypoint descriptors. The aim is to make the descriptor for the local image region highly distinctive while remaining as invariant as possible to variations such as change in illumination and 3D viewpoint.

The gradient magnitude and orientation are weighed by a Gaussian window $\sigma = 1.5$ as shown in figure 8. These samples are then accumulated in orientation histograms summarizing the contents over 4x4 subregions (right image).

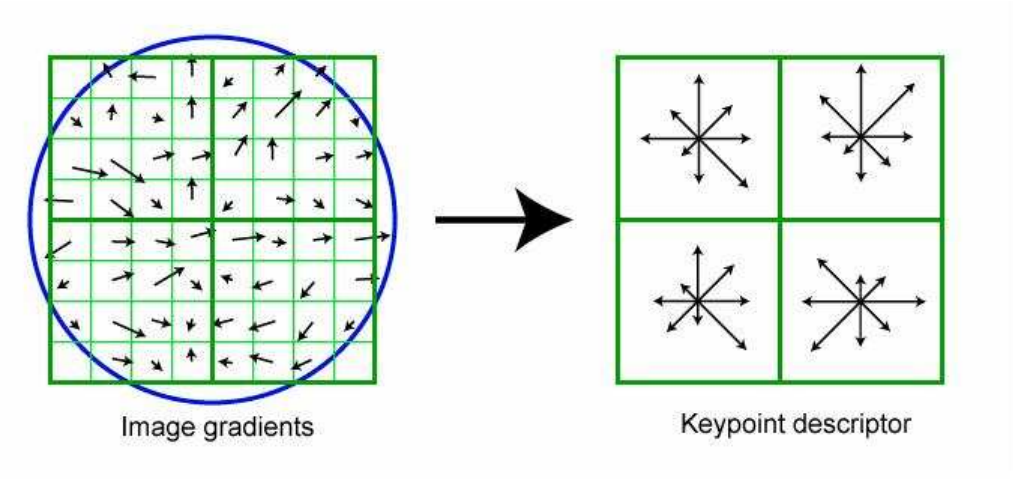


Figure 8: Computation of SIFT keypoint descriptor[26].

SIFT features are obtained with 4x4 arrays of histograms each with 8 orientation bins resulting to a 128 element feature vector for each keypoint (4x4x8). The resulting feature vector is modified by normalizing it to unit length to reduce the effect of illumination change.

3.4 SIFT feature matching

Using SIFT features for recognition purposes such as object and scene recognition requires that models of the objects to be recognized are stored in a database so that they can be used later for recognizing unknown samples.

Matching of keypoints is performed by finding the nearest neighbours of features from an unknown sample from features stored in a database of SIFT features extracted from training images. The nearest neighbour of a keypoint is defined as the keypoint with minimum Euclidean distance for the invariant descriptor vector [26].

In order for the match to be robust there needs to be a way to identify and hence discard features with incorrect matches in the training database. Examples of these features are those which arise from background clutter or were not detected in the training images.

The method proposed in [26] compares the distance of the closest neighbour to that of the second closest neighbour. Searching for a neighbour with minimum Euclidean distance can be very computationally expensive especially when the dimensionality of the feature space is high. According to [26], no algorithms are known that can identify the exact nearest neighbours of points in high dimensional spaces that are any more efficient than exhaustive search; for example, best algorithms such as the k-d tree provide no speedup over exhaustive search for more than about 10 dimensional spaces[13].

To overcome this problem [26] proposes an approximate method to be used. This method is called the Best-Bin-First(BBF) algorithm [4] which is a modification of the k-d tree algorithm. This method finds the closest neighbour with high probability.

The SIFT descriptor is highly distinctive and fast to compute, which makes it attractive for real-time applications.

3.5 SIFT implementations

Several implementations of SIFT are in existence. Most of these implementations are a result of the hard work of individuals who have decided to share their work with other researchers for use in their research work.

In the following sections some of these implementations will be surveyed.

3.5.1 David Lowe's SIFT implementation

David Lowe's implementation of SIFT [36] includes binaries for both Linux and Windows as well as Matlab scripts for performing SIFT keypoint detection and matching. The latest version at the time of writing is version 4, and includes the following utilities:

1. *Binaries for detection SIFT features* – There are two executables for this task namely *sift* and *siftWin32*. The former runs under Linux operating systems and the later runs under Windows. The syntax for these programs is as follows:

   ```
   $ sift -display <input_image> output_image
   ```

   This command detects keypoints in the input image *input_image* and displays them on the output image *output_image* specified in the command. In the Windows implementation the *sift* command is replaced by *siftWin32*. The input image must be in *pgm* format and the resulting output image is also in the same format. The resulting output image can then be viewed with any suitable image viewer, see figure 9 for an example.

   Alternatively the program can be instructed to output the keypoints to a file. This is useful in cases where the keypoints are going to be read and processed by other programs. The syntax for this is:

   ```
   $ sift <input_image> output_file
   ```

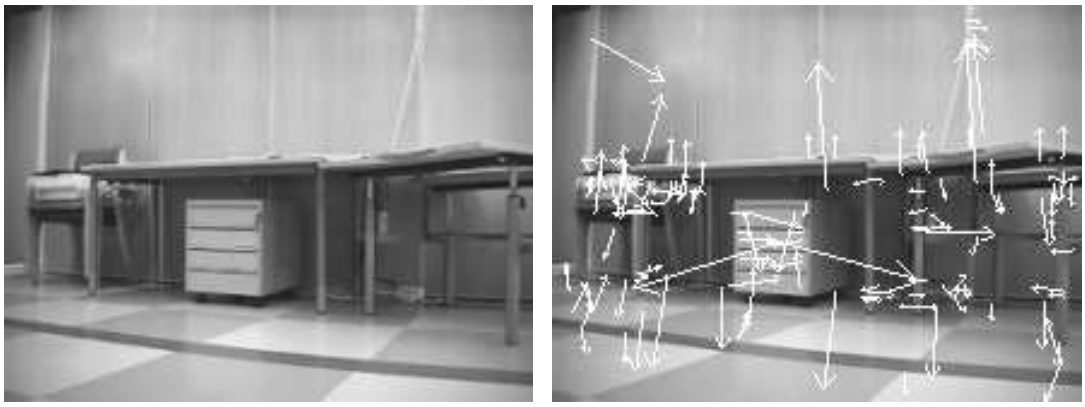


Figure 9: In this figure SIFT features detected in an image (right) are superimposed on the image(right) for visualization. The image resolution is 220 by 165 and the number of keypoints detected was 140. The arrows show the position, scale and orientation of the keypoints.

In this case the program will extract the keypoints from the image *input_image* and save them in the specified keypoint file *output_file*. The file is a normal text file whose format starts with two integers which specify the total number of keypoints and the length of the descriptor vector for each keypoint, in this case 128. Then follows the location of each keypoint in the image which is specified by 4 floating point numbers which give the subpixel row and column location of the keypoint, scale and orientation in radians. The orientation ranges from -PI to PI. Then lastly is the invariant descriptor vector for the keypoint which is made up of a list of 128 integers in the range $[0, 255]$.

2. *Binary for matching SIFT features* – Included in the implementation is also a program for matching keypoints from two images, this program is called *match*.The program operates on files containing keypoints, thus before using it you need to extract the keypoints from the images you want to match to files (see example above) and then invoke the program with the generated keypoint files and their corresponding images as parameters:

```
$ match -im1 image1 -k1 keyfile1 -im2 image2 -k2 keyfile2 > output
```

In the command above *image1* and *image2* are the two images whose keypoints are to matched while keyfile1 and keyfile2 are their corresponding keypoints re-

spectively. Out is the output image which contains the first image above the second and lines connecting the matching points. Figure 10 shows an example of the results of this program.

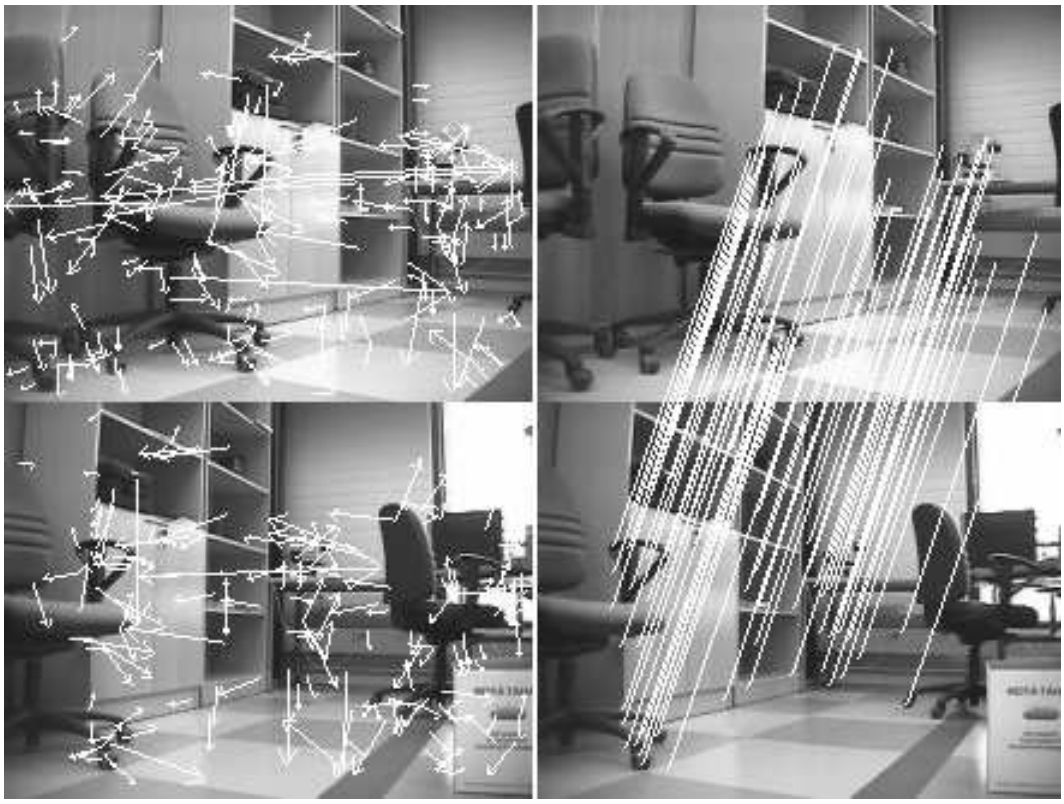


Figure 10: In this figure SIFT keypoints detected on two images(top left and bottom left) are shown together with an image showing the matches found between the keypoints in the two images connected by lines(top righ and bottom right)

3. *Matlab scripts* – Matlab scripts for loading SIFT features and finding matches between images are also supplied with the implementation. The script for script for extracting SIFT features is called *sift.m*. This script works by invoking the corresponding binary for extracting features and returns the features in a matrix form. The syntax for this command is shown below:

```
>> [output_image, descriptors, locations] = sift('input_image')
```

This command reads the image *input_image* and returns it SIFT keypoints. The returned values are:

(a) output_image – The image array in double format.

(b) descriptors – a K-by-128 matrix, where each row gives an invariant descriptor for one of the K keypoints. The descriptor is a vector of 128 values normalized to unit length.

(c) locations – a K-by-4 matrix, in which each row has the 4 values for a keypoint location (row, column, scale, orientation). The orientation is in the range $[-PI, PI]$ radians.

After extracting the keypoints with the *sift* command the showkeys command can be used to used to display the keypoints superimposed on the image. Its syntax is:

```
>> showkeys(image, locations)
```

The parameters *image* and *locations* are the outputs of the *sift* commands.

Finally, the match command matches keypoints from two images. It accepts as parameters the images to be matched, finds their SIFT features and displays lines connecting the matched points. Its syntax is:

```
>> match(image1,image2)
```

Where *image1* and *image2* are the two images to be matched.

This implementation is characterized by its speed of execution and simplicity. Working with the SIFT algorithm can be done with ease. In addition, the Matlab code provided does not depend on the image processing toolbox, adding to its flexibility.

3.5.2 Andrea Vedaldi's SIFT implementation

Another open implementation is that offered by Andrea Vedaldi [35]. Like Lowe's implementation, this implementation also comes with Matlab and C code. The code is split into several modules making it suitable for studying and understanding the algorithm. Worth noting is the fact that this implementation is not completely compatible, though very similar, to Lowe's.

It is available in source code form as Matalb m-files or as compiled MEX files. The later can be compiles under different platforms including Linux, Window and Macintosh.

1. *Computing and visualizing SIFT features* – This is done by executing a sequence of commands, first the features are detected and extracted using:

   ```
   >> [FRAMES,DESCRIPTORS]=sift(I)
   ```

   This will extract the SIFT frames FRAMES and corresponding descriptors DESCRIPTORS from the input image I. FRAMES is a 4xk matrix containing the frames whereby each frame corresponds to one column. The format of the frames is as follows:

   ```
   FRAMES(1:2,k)  center (X,Y) of the frame k,
   FRAMES(3,k)    scale SIGMA of the frame k,
   FRAMES(4,k)    orientation THETA of the frame k.
   ```

   DESCRIPTORS is a 128 by k array of SIFT descriptors containing one descriptor per column.

   The function can also return the Gaussian and difference-of-Gaussian scale sapace if required. This can be achieved with:

   ```
   >> [FRAMES,DESCRIPTORS,GSS,DOGSS]=sift(I)
   ```

   Where GSS and DOGSS are the Gaussian and difference-of-Gaussian scale spaces. The input image I is assumed to be grey-scale with in double storage class with the range normalized in [0,1].

2. *Visualizing results* – There are several Matlab functions available for displaying the results of computations. The most basic ones are `plotsiftdescriptor` and `plotsiftframe` for plotting SIFT descriptors and frames respectively. The former accepts the SIFT descriptors computed by the `sift` function while the later accepts the SIFT frames computed by the same. The syntax for these functions is as follows:

```
>> plotsiftdescriptor(DESCRIPTORS)
>> plotsiftframe(FRAMES)
```

3. `Matching SIFT features` – Matching can be done by `siftmatch` function. Its syntax is as follows:

```
>> MATCHES=siftmatch(DESCR1, DESCR2)
```

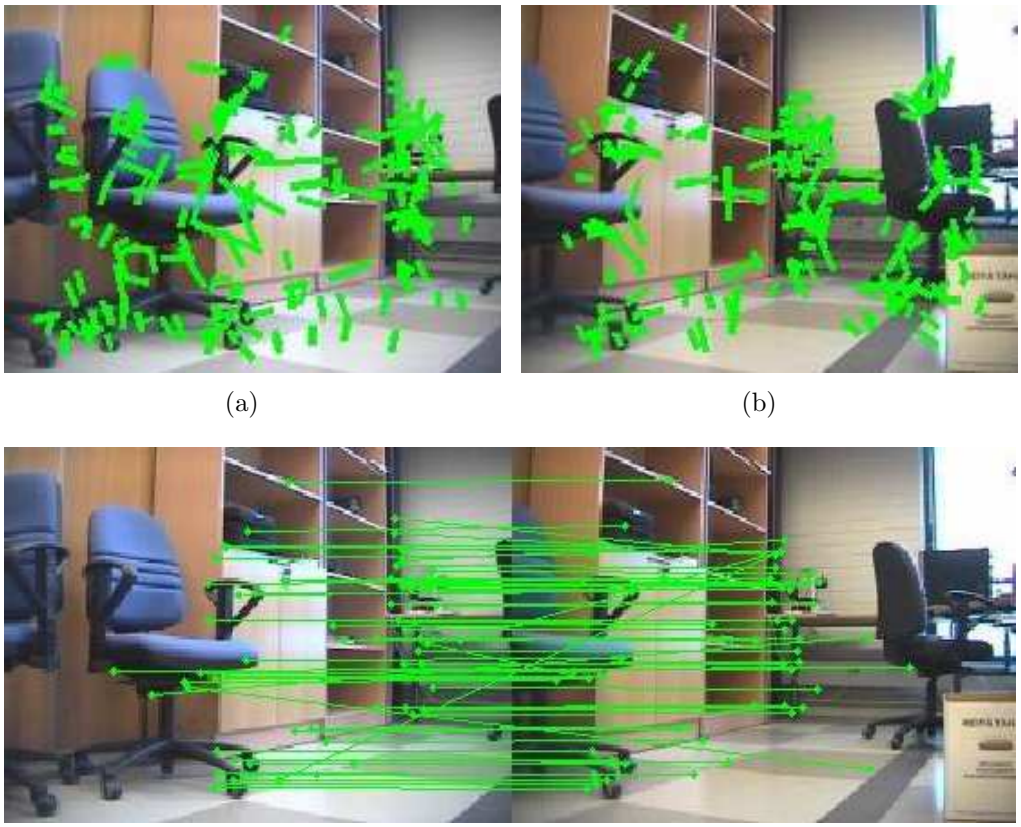
DESCR1 and DESCR2 are the matrices containing the SIFT keypoints to be matched. MATCHES specifies the indices of the matched keypoints in DESCR1 and DESCR2. Each column has two elements, the first gives the index of the first matched keypoint in DESCR1 and the second gives the index in DESCR2. The results of matching can be visualized using the `plotmatches` function as:

```
>> plotmatches(DESCR1, DESCR2)
```

The result of these functions are very similar to Lowe's with the exception that the `plotmatches` function can display matches in both color and grey-scale images.

Shown in figure 11 is an example of using these functions for SIFT keypoint detection and matching.

This implementation is very comprehensive and modular which makes its handy for learning and understanding how the SIFT algorithm works. In addition, it is well documented, which makes it easy to use.



(a) (b)

(c)

Figure 11: In this figure SIFT features detected in two images (above) are matched and the matches are shown connected by lines in the bottom image. 175 and 152 keypoints were detected in the two images (top left and right) and the number of matches found was 73.

3.6 SURF

Speeded Up Robust Features (SURF) [3] is a method for detecting and describing interest points.

The method is characterized by its rotation and scale invariance and it approximates previously proposed schemes, such as SIFT, with respect to repeatability, distinctiveness and robustness.

Unlike SIFT, which uses the Difference of Gaussian (DoG) function to approximate the scale space, SURF uses what are known as box type convolution filters for the

same purpose. It uses 9 x 9 box filters as approximations for Gaussian second order derivatives with $\sigma = 1.2$ to represent the lowest scale.

Implementing scale spaces normally requires the images to be repeatedly smoothed with a Gaussian filter and subsequently sub-sampled to build an image pyramid. Due to the use of box filters and integral images, SURF does not have to iteratively apply the same filter to the output of a previously filtered layer, instead it analyses the scale space by up-scaling the filter size. Higher scales are obtained by filtering the image with bigger masks.

Localization of keypoints in the image and over scales is done by applying non-maximum suppression in a 3 x 3 x 3 neighbourhood as in SIFT.

The keypoint descriptor is formed by first fixing a reproducible orientation based on information from a circular region aligned to the selected orientation and then constructing a square region aligned to the selected orientation from which the keypoint descriptor is extracted.

Two types of descriptors can be computed depending on how the region containing the keypoint is subdivided. The first kind is known as the short descriptor or SURF-36 and is computed from 3 x 3 subregions. The second type of descriptor is computed from 4 x 4 subregions and is known as SURF-128.

The resulting SURF descriptors are based on similar properties as the SIFT descriptor but with reduced complexity.

The SURF-36 descriptor is fast to compute and match but less accurate compared to the SURF-128 descriptor. The SURF-128 descriptor, on the other hand, is more accurate but slow in matching which makes it less interesting for application which depend on speed[3].

3.7 Rob Hess's SIFT library

This includes a collection of C functions for working with SIFT – computing and matching SIFT keypoints as well as executables for computing, matching and displaying keypoints. It also contains functions for computing image transforms from feature

matches using RANSAC[37]. The latest version at the time of writing is version 1.1.1. The library requires the OpenCV library for its operation. Some functions in the library require the GDK/GTK+2 and the Gnu Scientific Library (GSL).

There are two functions which computer SIFT features from images. The first one computes SIFT features using default parameter values and the second computes SIFT features using user-specified parameter values.

The first function has the following prototype:

```
int sift_features(IplImage*             img,
                  struct feature**      feat
                 )
```

Descriptions of the parameters for this function are given in 1 below. This function returns the number of features stored in the array `feat` or -1 if an error occurs.

Description 1 Parameters for the sift_features function.

img the image in which to detect features.

feat a pointer to an array in which to store detected features.

The second function has the following prototype:

```
int _sift_features(IplImage*            img,
                   struct feature**     feat,
                   int                  intvls,
                   double               sigma,
                   double               contr_thr,
                   int                  curv_thr,
```

```
    int                 img_dbl,
    int                 descr_width,
    int                 descr_hist_bins
)
```

Descriptions of the parameters for this function are given in 2 below. This function returns the number of features stored in the array $feat$ or -1 if an error occurs.

Description 2 Parameters for the _sift_features function.

---

img the image in which to detect features.

feat a pointer to an array in which to store detected features.

intvls the number of intervals sampled per octave of scale space.

sigma the amount of Gaussian smoothing applied to each image level before building the scale space representation for an octave.

contr_thr a threshold on the value of the scale space function $|D(\hat{x})|$, where $\hat{x}$ is a vector specifying feature location and scale, used to reject unstable features; assumes pixel values in the range [0, 1].

curv_thr threshold on a feature's ratio of principle curvatures used to reject features that are too edge-like.

img_dbl should be 1 if image doubling prior to scale space construction is desired or 0 if not.

descr_width the width, n , of the $n \times n$ array of orientation histograms used to compute a feature's descriptor.

descr_hist_bins the number of orientations in each of the histograms in the array used to compute a feature's descriptor.

Another class of functions included in the library is functions which deal with matching of keypoints. The first function is called `kdtree_build` and is used to build a k-d tree database from keypoints in an array. This function has the following prototype:

```
struct kd_node* kdtree_build(struct feature*  features,
                             int              n
                            )
```

Descriptions of the parameters for this function are given in 3 below. This function returns the root of a k-d tree built from `features`.

Description 3 Parameters for the kdtree_build function.

features an array of features.

n the number of features in features.

The second function in this class is called `kdtree_bbf_knn`. This function is used to find an image feature's approximate k nearest neighbors in a kd tree using Best Bin First search. This function has the following prototype:

```
int kdtree_bbf_knn(struct kd_node*      kd_root,
                   struct feature *     feat,
                   int                  k,
                   struct feature ***   nbrs,
                   int                  max_nn_chks
                  )
```

Descriptions of the parameters for this function are given in 4 below. This function returns the number of neighbors found and stored in `nbrs`, or -1 on error.

In addition to the functions the library also comes with a number of data structures which complement the functions. These include, for example, a data structures to represent an affine invariant image feature and a node in a k-d tree.

Description 4 Parameters for the kdtree_bbf_knn function.

kd_root root of an image feature kd tree.

feat image feature for whose neighbors to search.

k number of neighbors to find.

nbrs pointer to an array in which to store pointers to neighbors in order of increasing descriptor distance.

max_nn_chks search is cut off after examining this many tree entries.

4 System overview

The system is based on image matching, interest points from training images are stored in a feature database and are later used for localization by matching with features extracted from unknown images.

SIFT features are used as natural landmarks. The reason for choosing SIFT features as natural landmarks is mainly due to their stability and robustness, invariance to image transforms such as scale and illumination as well as changes in 3D camera viewpoint[26].

Each node is represented by a collection of SIFT features stored in a feature database.

The system comprises of three main modules. These are:

- Training module
- Localization module
- Database module

The database module keeps and organizes information about nodes and features. The training module operates by updating the database with features that are extracted from images taken from the nodes. The localization module, on the other hand, operates by querying the database and comparing the information it obtains from the database to that obtained from the images taken from the nodes, the results of this comparison are used for localization.

Figure 12 shows the modules in the system and the interaction between them.

4.1 The training module

The training module is responsible for the learning of the robot. It enables the robot to learn by taking images of a node and extracting features from the images. These

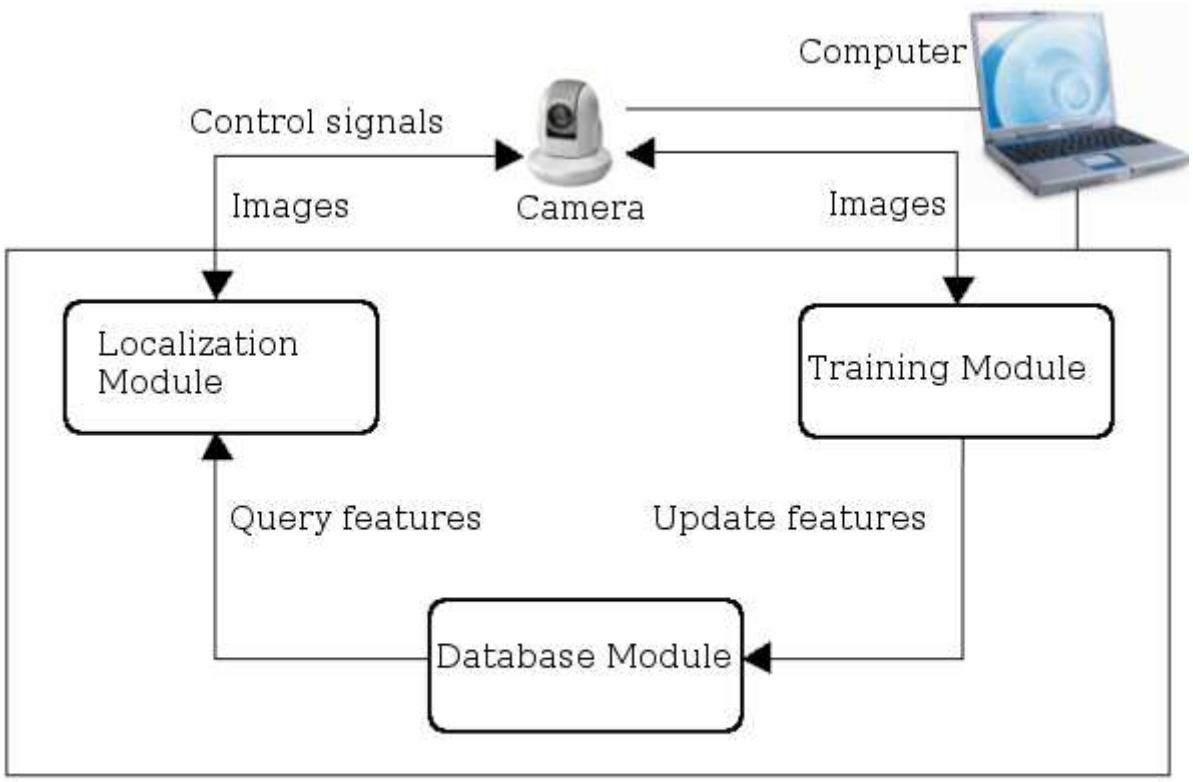


Figure 12: Block diagram of the system showing the interactions between modules.

features are then stored in a database ready for use in localization.

Algorithm 3 summarizes the operation of the training module.

The operation begins by the module receiving the ID of the node for which to train. This information is used by the module to distinguish the nodes when updating the information in the database.

After the node ID has been received the module moves the camera to its home position. This position is used as a reference position.

From the reference position the camera is repeatedly moved to the left at approximately 50 degrees per move until it reaches the maximum pan angle in the left direction. At

Algorithm 3 Operation of the training module

1. $T \Leftarrow 25$ {Minimum allowed number of features per image}
2. $t \Leftarrow 2$ {Time to wait for the camera to stop moving}
3. Pan camera to home position

4. **repeat**
5. Capture an image from the camera
6. Extract SIFT features from the image
7. **if** The number of features found $\geq T$ **then**
8. Store the features to the database
9. **else**
10. Ignore the image
11. **end if**
12. Pan camera to the right
13. Wait t seconds for the camera to stop
14. **until** The camera can not pan to the right

15. Pan camera to home position
16. Wait t seconds for the camera to stop

17. **repeat**
18. Pan camera to the left
19. Wait t seconds for the camera to stop
20. Capture an image from the camera
21. Extract features from the image
22. **if** The number of features found $\geq T$ **then**
23. Store the features to the database
24. **else**
25. Ignore the image
26. **end if**
27. **until** The camera can not pan to the left

each move there is a pause of a 2 seconds to let the camera stop completely before any image is captured.

When the camera stops an image is captured and SIFT features are extracted from the image. If the number of features found is greater than 25 the image is considered informative and the features are stored into the database, otherwise the image images is considered less informative and ignored. The cut-off value of 25 features per images was selected to be the same as the average number of matches in training images.

When the maximum pan angle to the left is reached the camera is moved to the home position again and the procedure is repeated for the right half, see figure 13.

Figure 13: Image of camera with embedded sketch showing the camera configuration used during training.

The operation of the training module can be in two modes. In the first mode the operator moves the robot from one position to another within a node. At each position

the module captures training images, extracts features and stores the features into the database. In this case the operator decides which positions in a node give maximum coverage of the node.

In the second mode the operator can just drive the robot around a node and the module continuously captures images, extracts features and stores the features into the database.

The first approach has the advantage of giving full control of the training process to the user by allowing him or her to choose which positions within a node are used for training, which also enables the user to control the number of features entered into the database for each node.

The second approach, on the other hand, has the advantage of being less involving as the user has very little interaction with the system. In this approach, however, the number of features stored into the database becomes unnecessarily very large which has the effect of slowing down the system.

4.2 The localization module

The localization module is responsible for the actual localization of the robot.

It operates by capturing images of the scene from the camera, extracting features and then finding the closest matching image from the feature database based on the number of keypoint matches on the images.

Algorithm 4 summarizes the operation of this module.

The operation involves capturing an image of the scene from the camera and extracting SIFT features from the image. As in the training module, if the number of features found is less than 25 the image is ignored and another image is captured after turning the camera to the right or to the left by approximately 50 degrees.

When enough features have been found from an image a worker thread is created for each node. The thread finds the closest match to the images in the given node. Matching of keypoints is based on Euclidean distance between the keypoints in the

Algorithm 4 Operation of the localization module

1. Initialize t {Time to wait between localization attempts}
2. Initialize N {The number of nodes}
3. $T \Leftarrow 25$ {Minimum allowed number of features per image}
4. Pan camera to home position

5. **repeat**
6. Capture an image from the camera
7. Extract SIFT features from the image
8. Pan camera to the right
9. **until** The camera can not pan to the right Or number of features found $\geq T$

10. **if** The number of features found $\leq T$ **then**
11. Pan camera to home position
12. **repeat**
13. Pan camera to the left
14. Capture an image from the camera
15. Extract features from the image
16. **until** The camera can not pan to the left Or number of features found $\geq T$
17. **end if**

18. **if** The number of features found $\leq T$ **then**
19. exit
20. **end if**

21. Create N work threads
22. Wait for the threads to finish execution
23. Find the maximum value from thread return values

test image and those in the model image. The best match of a keypoint is a keypoint which has the shortest Euclidean distance to the keypoint.

In order to eliminate ambiguous matches two things are done. First each keypoint in the test image is matched against all the keypoints in a model image and then the process is reversed by matching each keypoint in the model image against all the points in the test image. Second, in order for a keypoint to qualify as a consistent match it has to satisfy the following criteria

$$\left(\frac{dist(x, x_{best})}{dist(x, x_{second\,best})}\right)^2 < t \tag{13}$$

which requires that the ratio of the squared ratio of Euclidean distances between a keypoint and its nearest neighbour to that between a keypoint and its second nearest neighbour be less than a threshold t. The value of t which gave good performance is 0.49, thus all matches which give a value of t greater than 0.49 are discarded. The process of finding the best match of an image within the worker threads is summarized in algorithm 5.

Each thread is given the test image and the ID of the node it is working on. Using the ID the thread retrieves the list of all the images which belong to that node. After retrieving the list of images to process the thread repeatedly retrieves an image from the database while matching the test image against the retrieved image(forward matching) and the retrieved image against the test image(reverse matching).

Searching of nearest neighbour of a keypoint is done by building and searching a k-d tree of the features of the test image and that of the model image. A k-d tree is built once for the test image and at each iteration a k-d tree of a model image is built. The reason why a k-d tree is build for each model image is because of the memory constraints. The SIFT keypoint descriptor consists of 128 integers. In a 32 bit machine this requires 512 bytes of memory. Thus, for a database of 900,000 keypoints, as the one used in the experiments this requires more than 400Mb of memory:

$$\frac{512\,bytes * 900000}{(1024)^2} = 439.45Mb. \tag{14}$$

When a thread finishes execution it returns a score which is twice the minimum of forward and reverse matches found in the best match. The scores from all the threads are collected in the main thread and the maximum score is found. If the maximum score is greater than a threshold value the module assumes the current location to be the location that corresponds to the thread which returned the maximum score. If the maximum score found is less than the threshold the result is considered uncertain and the localization process is repeated from the beginning. The threshold value used for this purpose is 20, which is was determined experimentally.

Algorithm 5 Operation of a worker thread

1. Get node ID {ID of the node to work on}
2. Get test image {The image of the scene}
3. $N \Leftarrow 0$ {Number of images to process}

4. $max \Leftarrow 0$ {maximum score}
5. $score \Leftarrow 0$ {Matching score}
6. $img\_ids[\,] \Leftarrow$ empty {Set list of image IDs to empty}
7. Connect to database
8. Retrieve a list of images to process
9. $img_ids \Leftarrow$ list of image IDs to process
10. $N \Leftarrow$ number of images to process

11. Build k-d tree of features in test image

12. **for** $i = 0$ to N **do**
13. $mf \Leftarrow 0$ {Number of forward matches}
14. $mr \Leftarrow 0$ {Number of reverse matches}
15. Retrieve an image with ID $img\_ids[i]$
16. Build k-d tree of features in model image
17. Match test image against model image {Forward matches}
18. $mf \Leftarrow$ number of forward matches
19. Match model image against test image {reverse matches}
20. $mr \Leftarrow$ number of reverse matches
21. **if** $mf < mr$ **then**
22. $score = 2 * mf$ {Find the minimum}
23. **else**
24. $score = 2 * mr$
25. **end if**

26. **if** $score > max$ **then**
27. $max = score$ {Update maximum score}
28. **end if**
29. **end for**

30. **return** max
31. **return** node ID

4.3 The database module

The database module serves the purpose of storing and organizing information about nodes and the features extracted from them.

It consists of a database with three tables, these are as follows:

- NODE table – This table keeps information about nodes, the information includes node ID and node description.
- IMAGE table – This table keeps information about training images for each node. This information include image ID and the number of features which were found in the image.
- FEATURE table – The feature table keeps information about SIFT features for each nodes' images. Each row has room for 128 values of SIFT keypoint descriptor together with other identifying information.

Figure 14 shows the tables in the database module and the relationships between them.

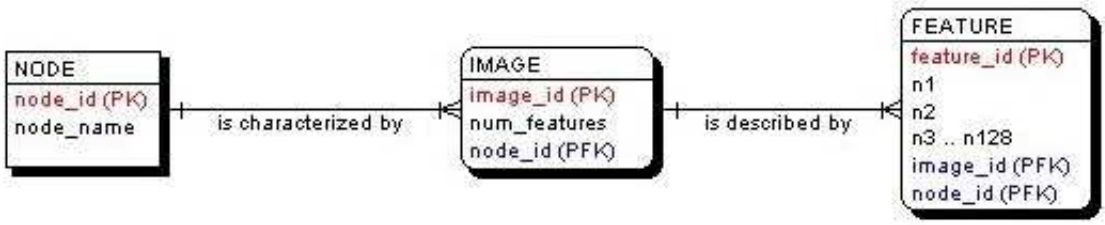


Figure 14: Tables in the database and their relationships.

4.4 Software tools

The following sections describe various software tools which were used in the process of developing the system and how they were used.

4.4.1 MySQL

MySQL is a very popular open source relational database management system characterized by high reliability, performance and ease of use. The fact that MySQL runs on multiple platforms, including Linux, Windows and OS/X makes it the database of choice for a variety of applications ranging from web applications to specialized embedded applications.

In this system MySQL is used for storing and retrieving information about nodes and SIFT features.

4.4.2 OpenCV

OpenCV is an open source computer vision library whose development is sponsored by Intel. It is intended for use by both commercial and non-commercial users including researchers, commercial software developers, governments and camera vendors.

It consists of C functions which implement standard computer vision algorithms such as edge and corner detection and motion analysis. Other features include linear algebra and raw matrix support, windowing system and functions and datastructures for reading and writing and images video streams.

In this project OpenCV is used for loading images from disk files ready for extracting features.

4.4.3 libCURL

libCURL is a library for transferring data with a URL syntax which supports a wide array of protocols including FTP, FTPS, HTTP, TELNET and LDAP among others. It is reliable, easy to use and portable – it runs on multiple operating systems including Linux, Windows, Solaris and FreeBSD.

libCURL has been used to send CGI control commands to the camera, these include commands to pan the camera as well as retrieving images from the camera.

4.4.4 Rob Hess's SIFT library

A discussion of this library is given in section 3.7. This library has been used for computing and matching keypoints in the final implementation of the system. The reason why this library has been chosen is because it is very comprehensive – it has functions to perform all the operations need in the system. Additionally, the source code for the library is also availabe which makes it possible to customize the library.

5 Technology evaluation

This section describes technology evaluation experiments which were carried out with the SIFT algorithm in order to study the matching of SIFT features, that is, determining when failure occurs, what causes failure and how to overcome it; and to determine a suitable discrete classifier which is based on SIFT keypoint matches between two images.

5.1 Test data

Test data for these experiments consisted of JPEG images which were taken from the test environment using the on-board camera of the robot. Each image had a resolution of 320 by 240 pixels. The images were taken with some degree of overlap between them enough to allow successful matching of SIFT keypoints between them.

The number of test images used for each node is shown in table 1.

Node	**Number of images used**
N1	99
N2	69
N3	130
N4	100
N5	70
N6	78
N7	85

Table 1: Number of training images used for each node

The test images were converted to greyscale (`pgm` format) and given numeric names to save storage space and simplify processing respectively. Thus for each node the image names ranged from `1.pgm` to `N.pgm` where N is the number of test images in a particular node.

5.2 Causes of failure in matching SIFT features

The aim of this experiment was to identify the cases in which the SIFT algorithm fails with test images and determine the cause of failure.

The experiments were carried out using David Lowe's implementation of SIFT [36]. Every image was matched against the rest of the images and the number of matches as well as the number of keypoints found in the matched images were recorded. Each image was classified to belong to a node in which the maximun number of matches occurred. Keypoint matching was done according to a scheme proposed in [36].

For each node the misclassified images were noted and analysed.

Results and analysis

Classification of the images was based on the number of matches between the images – the test image was assumed to be belonging to the node which gave the highest number of matches.

The classification results are summarized in table 2.

Node	Number of images	Images missclassified	Percentage error
N1	99	3	3.03
N2	69	8	11.59
N3	130	19	14.61
N4	100	4	4
N5	70	12	17.14
N6	78	22	28.2
N7	85	10	11.76
Total	631	78	

Table 2: Summary of classification results

Analysis of results showed that there are three main causes of failure when matching features between images. These are:

1. *Lack of good overlap between images* – SIFT finds similar points in the images which are being matched. If there is no good overlap between the images the algorithm is likely to fail or produce incorrect results. Results showed that if there is good overlapping in the images most of the matches found are correct. Figure 15 shows an example of two images with good overlapping between them and the matches found between the images.

2. *Presence of similar objects* – Presence of similar objects such as trash cans, furniture and tube lights confuses the algorithm which leads to incorrect matches. Similarly, bright spots which are due to reflections lead to incorrect matches where by many keypoints are matched to a single incorrect keypoint. An example of this phenomenon is shown in figure 16.

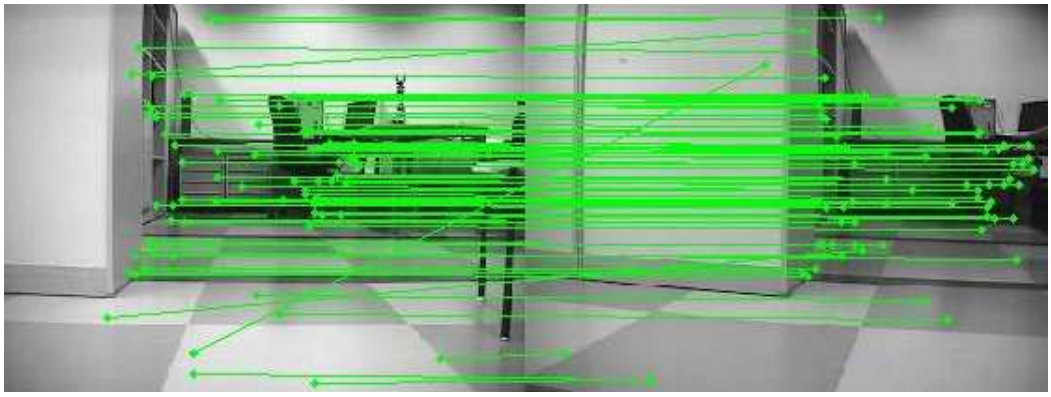
3. *Insufficient features in images* – If the number of features found in one of the images which are being matched is very low the result of matching become uncertain. Figure 17 shows examples of images with very low number of features.

The problem of lack of good overlap between images appeared across all the nodes and was more frequent than the others. This can be attributed to the fact that the training images were taken manually by turning the robot around. The problem of insufficient features in training images had been common mostly in the corridors (N2,N5 and N7) where there are plenty of plain walls. Of all the three problems the presence of similar objects was less common. This can be attributed to the fact that similar objects across the nodes do not have the same surrounding objects and orientations. For example a bookshelf in one node may be full of books and placed against a wall while a similar bookshelf in another node is partially filled with books and placed in the middle of the node. In this scenario even though the two bookshelves are the same it is less likely to confuse one with the other.

(a) First image

(b) Second image

(c) Forward matching: 74 matches



(d) Reverse matching: 83 matches

Figure 15: An example of good overlapping between training images – most of the matches remain after reversing the matching.

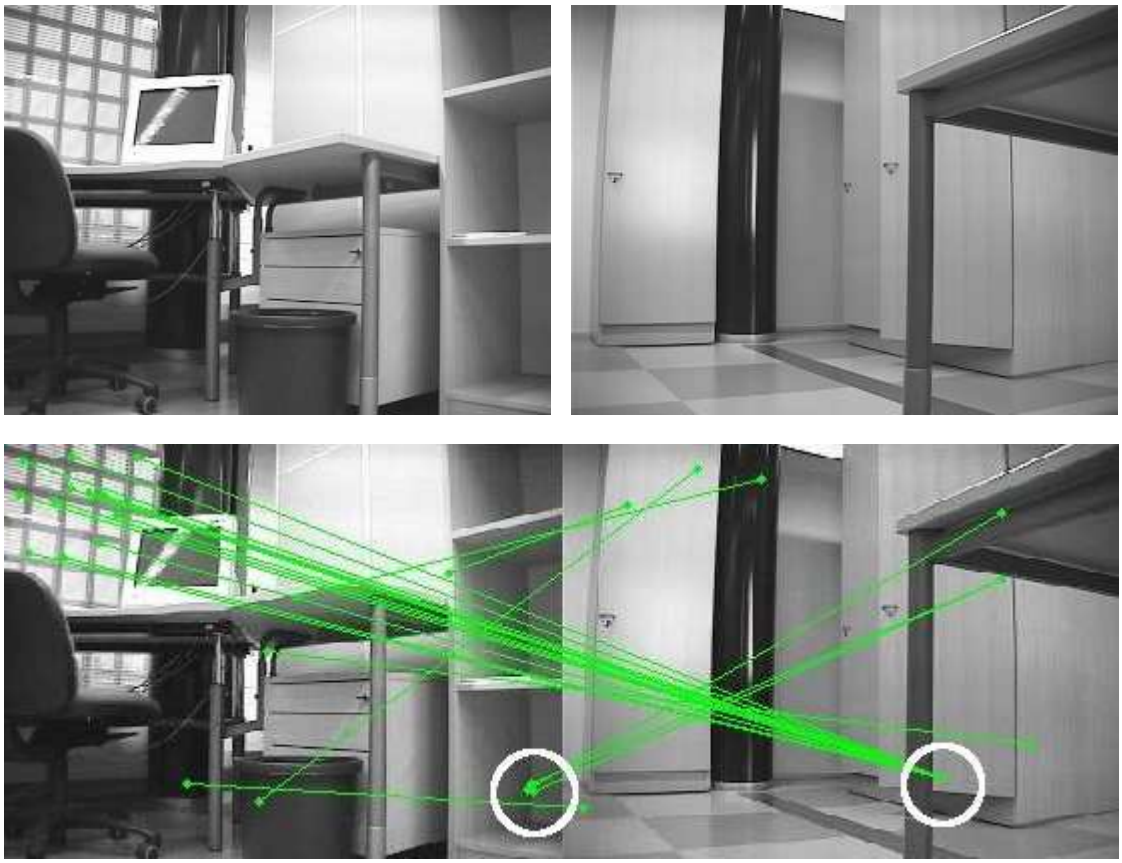


Figure 16: An example of incorrect matches – many keypoints are matched to a single incorrect one.

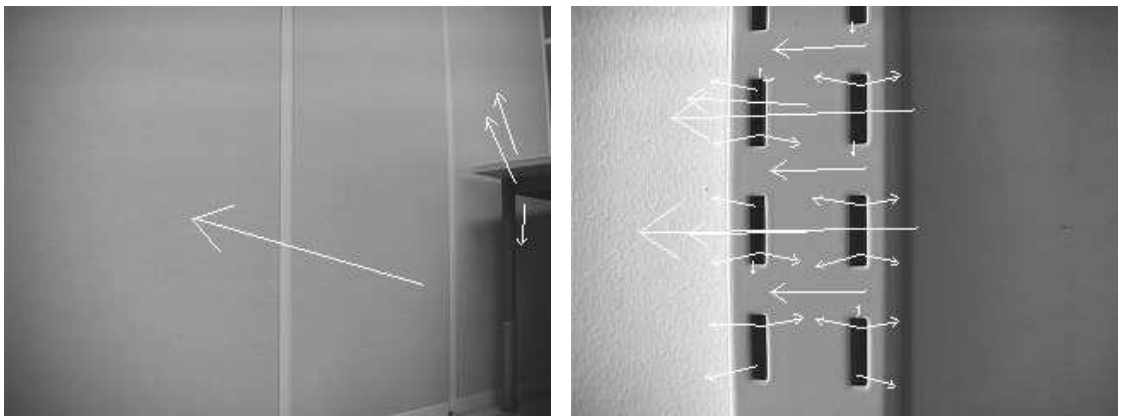
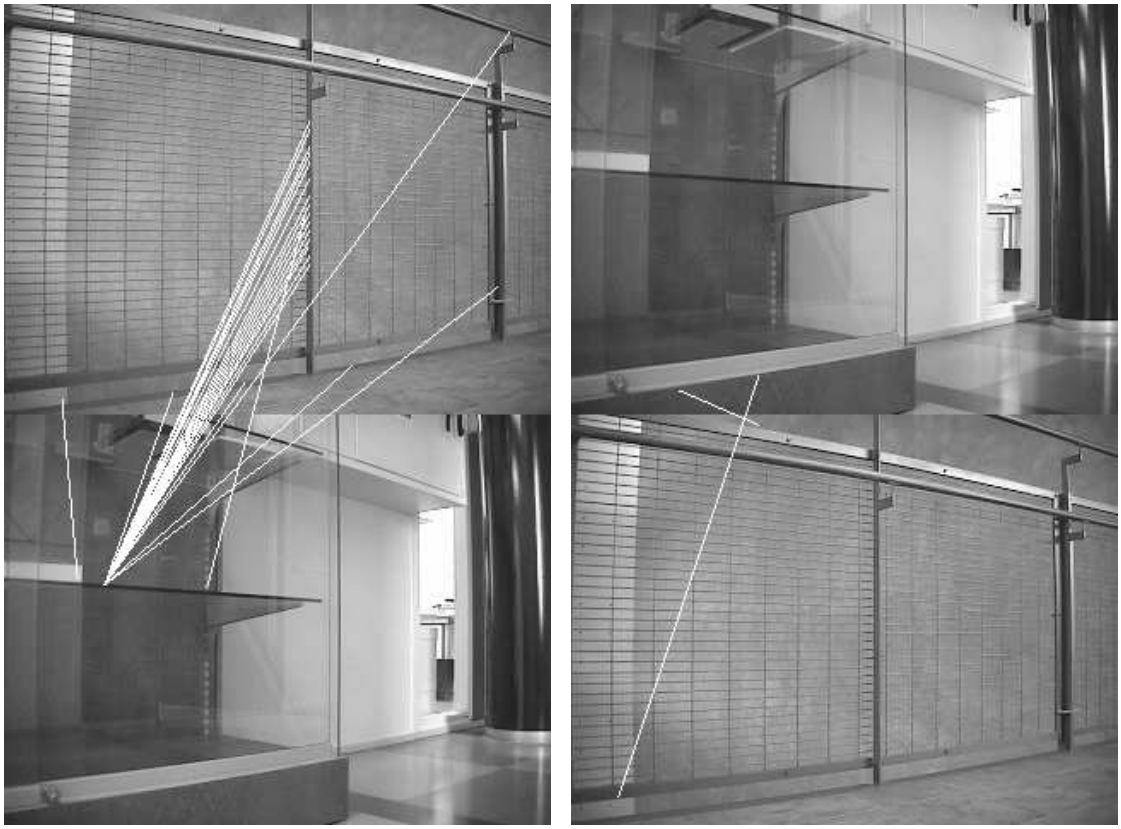


Figure 17: Examples of images with small number of features.

Results suggest that if there is enough overlap in the training images the matching becomes more reliable and correct. Experiments also showed that matching the images in reverse helps to detect failures by comparing the number of matches found in the forward and reverse matching. This is because correct matches remain even if the matching is reversed whereas the wrong ones vanish(see figure 18).



(a) Forward matching (b) Reverse matching

Figure 18: Example of a large difference between features found in forward and reverse matching.

The problem of low number of features, on the other hand, can be overcome by the use of multiple measurements, that is, if the number of features extracted from an image is below a chosen threshold value the image is ignored and another image is taken.

5.3 Classifier based on SIFT keypoint matches

The aim of this experiment was to design a measure or numeric score that would be used as a classifier. This measure will serve as a cut-off value when matching images, that is, when matching a test against model images the best match to the test image is considered the one which gives the largest value of this measure.

To achieve this goal different ways of combining the number of matches of keypoints were studied. The following measures were used to classify the test images, these give a metric value which is used as a classifier.

- Sum of matches in a pair of images
- Maximum of forward matches and reverse matches
- Twice the minimum of forward and reverse matches

The classification was carried out using the leave-one-out validation criteria, that is, for each node every image was set aside in turn and used as test image. This image would then be matched against all the images in all the nodes. The test image was assumed to have been from the node which gave the larges value of the given score.

The performance results of these classifiers are shown in figure 19.

Results and analysis

From the ROC graph it can be seen that the classifier which is based on twice the minimum number of matches performs better than the others. This fact can be attributed

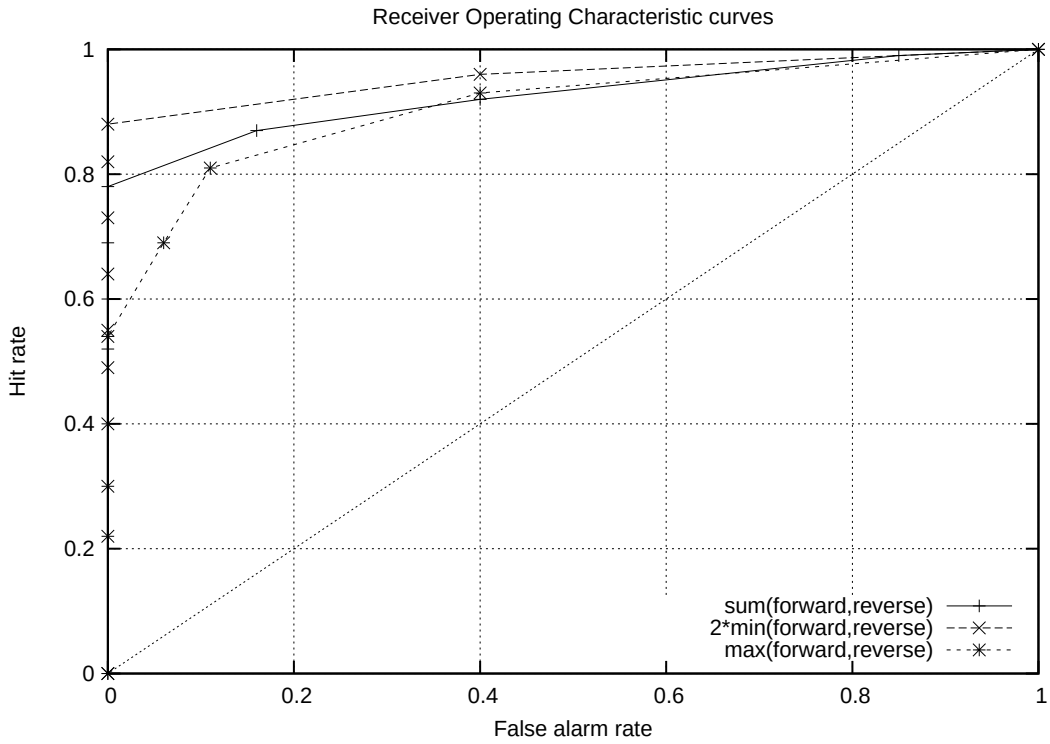


Figure 19: ROC curves showing performance of classifiers.

to the two way matching of images. For most of wrong matches disappear when the matching is reversed, it means if dissimilar images are matched and the minimum number of matches taken the score will be very low because of the big difference between the forward matches and reverse matches. On the other hand, if similar images are matched most of the matches will remain after reversing the matching. This gives a higher value for the score.

6 System evaluation

This section describes experiments which were carried out in order to test the final implementation of the system.

6.1 Test data

The test data for these experiments consisted of a database of varying number of SIFT features. The features in the database were extracted from test images which were taken randomly from the test environment.

6.2 Effect of average number of features per node on localization accuracy

The aim of this experiment was to study how the localization error varies with the average number of features in the database per node. The number of features in the database for each node was gradually increased while making localization attempts at each increase. For each level of features 10 localization attempts were randomly made and the percentage error computed.

Results and analysis

Results for this experiment are summarized in figure 20. It can be seen that the percentual localization error decreases with increasing average number of features per node in the feature database.

From figure 20 it can be seen that the error rate varies rapidly with the average number of features per node when the average number of features per node is less than 25,000, beyond this value the change seem to be gradual. Also the graph suggests that in order to achieve an error rate of less than 10% there need to be at least 35,000 features per node.

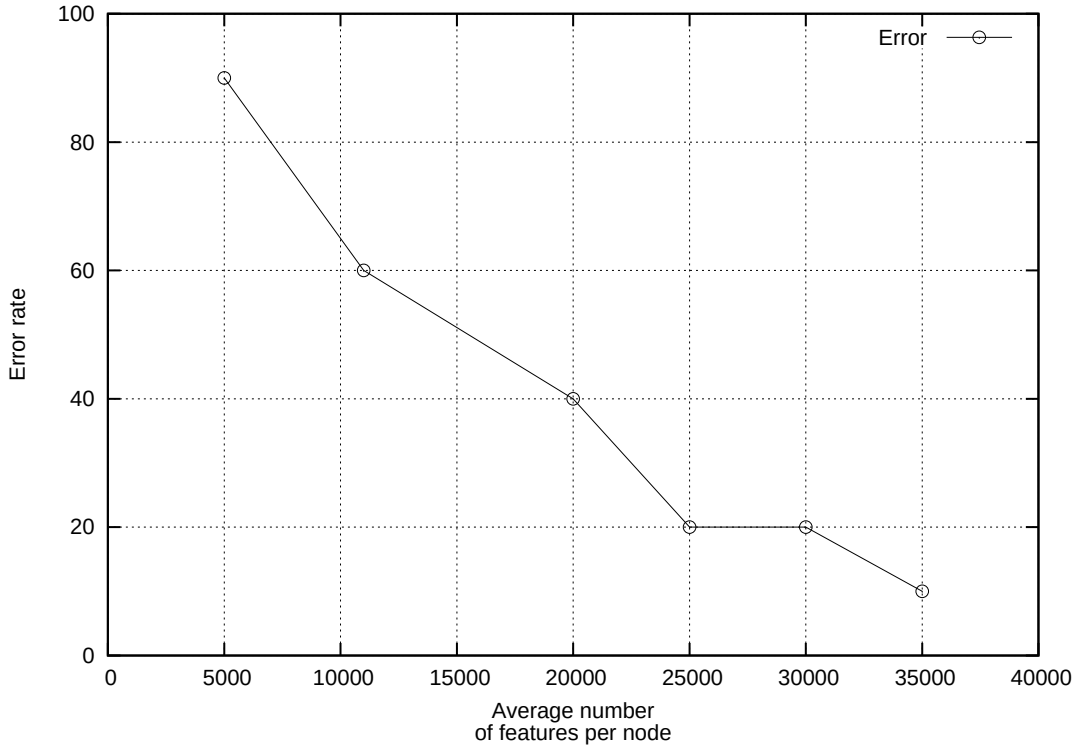


Figure 20: This graph shows how localization error varies with the average number of features per node in feature database.

6.3 Effect of number of features on localization time

This experiment was designed in order to study how localization time is affected by number of features in the feature database. This was done by varying the size of the database and record the wall-clock time taken to complete the localization process. The experiment was done using IBM Thinkpad laptop computer with 512Mb of memory and a 1.5Ghz Pentium M processor running the Debian GNU/Linux operating system.

Results and analysis

Results for this experiment are summarized in figure 21. The plot shows that localization time increases in an approximately exponential manner with the average number of keypoints in the database. This is an expected behavior because the bigger the number of features to match the longer it takes to search for the best match for an

image. The reason for this is the time to query the feature database in order to retrieve the features also increases. It should, however, be noted that the time to search for a nearest neighbor of a keypoint is not affected by the average number of features per node in the features database, rather it is affected by the average number of keypoints per image. This is due to the way the matching of images is done, that is, an image is retrieved from the database, a k-d tree of the features in the image is built and matching is done between the keypoints in this k-d tree and that of a test image, this process is repeated for all the images in the database.

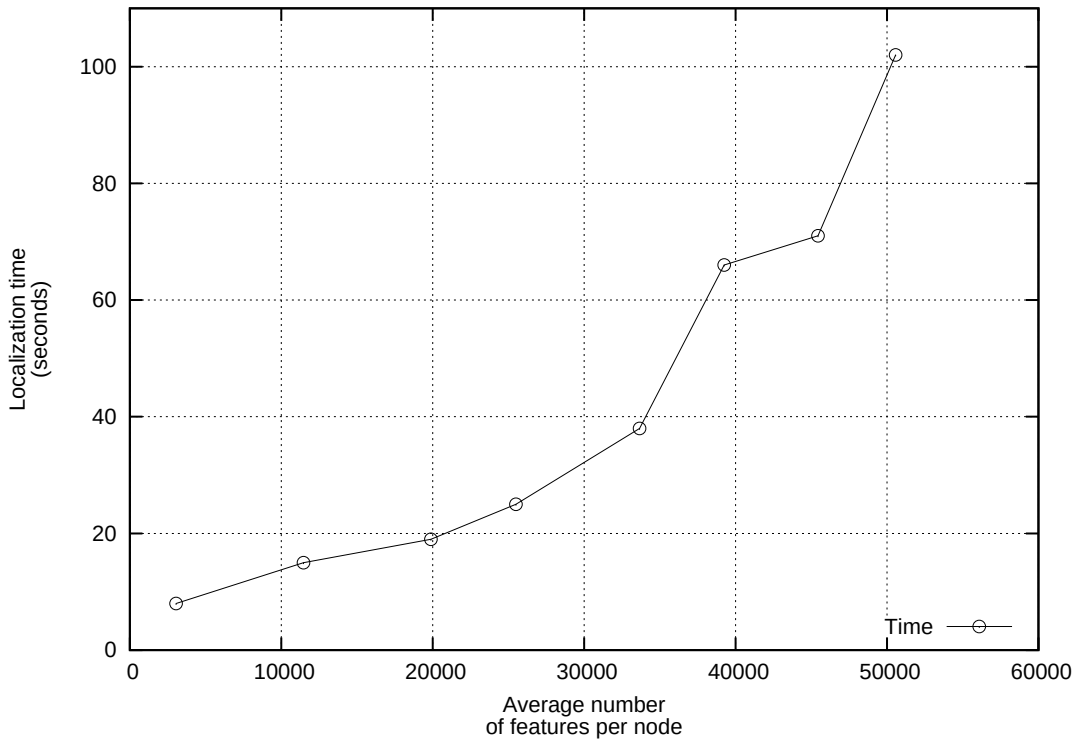


Figure 21: This graph shows how localization time varies with the average number of features per node in feature database.

7 Conclusions and future work

A mobile robot localization system which uses SIFT image features as natural landmarks has been presented. SIFT features extracted from training images were stored in a feature database during training phase and during localization an unknown image of the scene was taken and features extracted from it were matched against the features stored in the feature database.

To detect failure and reduce the error rate it was necessary to match SIFT features in two ways–forward and reverse. If there was a significant change in the number of matches found between forward and reverse matching, or if the number of matches found was very low, the result was considered uncertain and the localization process was repeated.

Matching of keypoints was accomplished by creating a separate thread for each node. The thread searched for the best match of a test image in a given node by loading and matching one image at a time. At the end when all threads have finished running the overall best match was found. This approach was adopted because the number of keypoints was very large, thus it would require more memory than an average PC can offer.

Although the matching method adopted decreased the error rate significantly, errors still remained that were mainly caused by high degree of similarity between the nodes. Experiments showed that the error rate tended to decrease with increasing number of training features used in the database but this had a negative effect on localization time.

The localization system presented in this thesis has some important similarities and differences with the approaches presented in the literature. The most important similarity is the use local invariant image features as natural landmarks, examples include [49] and [5]. In the former Harris-Laplace interest points are indexed in two separate databases whereby one of the databases is used for coarse localization and the other for fine localization. In the latter SIFT features were used to build a 2D map of the environment for metric localization.

Although both the approach presented in [49] and the one used in this thesis used local

invariant image features the methods to match images are different. The former used the Vector Space Model (VSM) to accelerate the matching process, which is inspired by text retrieval techniques, in contrast to the k-d tree used for the same purpose in this work.

The overall performance of the system was encouraging and therefore the work may be of interest for further research in the future whereby, among other issues, the following should also be addressed:

1. Use of constraints in matching of features – The use of constraints such as geometric ones will help make the system more robust by detecting and eliminating false matches due to high degree of similarity between nodes.

2. Active camera control – Currently when the number of features found in an image is very low the system moves the camera in a pre-determined way so as to look for more features before proceeding with the localization process. Active control of the camera, that is, moving the camera to the direction where there appears to be more features will have the advantage of reducing the time it takes to get enough features for localization.

3. Speech recognition interface – This will enhance the human-robot interaction by enabling the robot to understand simple voice commands. For example, its would be interesting to have the robot understand and execute commands such as "Go to the machine vision lab", or be able to answer to questions like "Where are you now?".

4. Active search – Currently, if there are no enough features for localization in the current view the system only turns the camera while the robot stays stationary. Active searching can help increase the system robustness by allowing the system to move the robot from one point to another in search for more features.

REFERENCES

[1] S. Aoyagi, Y. Kiguchi, K. Tsunemine, and M. Takano. Position and orientation measurement of a mobile robot. *Transactions of IEE Japan*, 121-c(2):375–384, 2001.

[2] J.M. Armingol, L. Moreno, A. de la Escalera, and M.A. Salichs. Landmark perception planning for mobile robot localization. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 3425–3430, May 1998.

[3] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded Up Robust Features. In *9th European Conference on Computer Vision*, pages 404 – 417, Graz, Austria, 2006.

[4] J Beis and D.G Lowe. Shape indexing using approximate nearest-neighbour search in high dimensional sapaces. In *In Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 1000 – 1006, 1997.

[5] Maren Bennewitz, Cyrill Stachniss, Wolfram Burgard, and Sven Behnke. Metric localization with scale-invariant visual features using a single perspective camera. In *Proceedings of European Robotics Symposium*, pages 195 – 209, Pacermo, Italy, 2006.

[6] W. Burgard, D. Fox, and D Hennig. Fast grid-based position tracking for mobile robots. In *Proceedings of the 21th German Conference on Artificial Intelligence, Germany*, 1997.

[7] E. R. Davies. *Machine Vision: Theory, Algorithms, Practicalities.* Morgan Kaufman, San Francisco, 3rd edition, 2005.

[8] G.N. Desouza and A.C. Kak. Vision for mobile robot navigation: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(2):237–267, February 2002.

[9] M. W. M. Gamini Dissanayake, Paul Newman, Steven Clark, Hugh F. Durrant-Whyte, and M. Csorba. A solution to the Simultaneously Localization and Map building (slam) problem. *IEE Transactions on Robotics and Automation*, 17:229–241, 2001.

[10] Tom Fawcett. ROC graphs: Notes and practical considerations for researchers. *Kluwer Academic Publishers*, March 2004.

[11] D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999.

[12] Prof. Bill Freeman. Lecture notes in computational photography. web document (referred on 28.3.2007). http://groups.csail.mit.edu/graphics/classes/CompPhoto06/html/lecturenotes/matchingApril112006.ppt.

[13] Jerome H. Freidman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209 – 226, 1977.

[14] J.-S. Gutmann, W. Burgard, D. Fox, and K. Konolige. An experimental comparison of localization methods. In *Proc. of the IEEE/RSJ InternationalConference on Intelligent Robots and Systems*, 1998.

[15] Jens-Steffen Gutmann. Markov-kalman localization for mobile robots. In *IEEE International Conference on Pattern Recognition*, volume 2, pages 601–604, 2002.

[16] Jens-Steffen Gutmann and Dieter Fox. An expririmental comparison of localization methods continued. In *IEEE/RSJ International Conference on Intelligent Robotis and Systems*, pages 454–459, EPFL, Lausanne, Switzerland, October 2002.

[17] C. Harris and M. Stephens. A combined corner and edge detection. In *Proceedings of The Fourth Alvey Vision Conference*, pages 147–151, 1988.

[18] J.B. Hayet and M. Lerasle, F.; Devy. A visual landmark framework for indoor mobile robot navigation. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 3942–3947, May 2002.

[19] John M. Holland. *Designing Autonomous Mobile Robots*. Elsevier Inc., 2004.

[20] Patric Jensfelt. *Approaches to Mobile Robot Localization in Indoor Environments*. PhD thesis, Royal Institute of Technology (KTH), Sweden, 2001.

[21] William J. Raynor Jr. *The International Dictionary of Artificial Intelligence*. The Glenlake Publishing Company, Ltd., March 2000.

[22] Michael Kofler. *The Definitive Guide to MySQL*. Apress, 2nd edition, 2004.

[23] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu.

[24] Weiguo Lin, Songmin Jia, Takafumi Abe, and Kunikatsu Takase. Localization of mobile robot based on ID tag and WEB camera. In *IEEE Conference on Robotics, Automation and Mechatronics*, volume 2, pages 851–856, December 2004.

[25] David G. Lowe. Object recognition from local scale-invariant features. In *IEEE International conference on computer vision*, volume 2, pages 1150–1157, September 1999.

[26] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Internatonal Journal of Computer Vision*, 60(2):91–110, January 2004.

[27] Vladimir J. Lumelsky. *Sensing, Intelligence, Motion–How robots and humans move in an unstructured world*. John Wiley and Sons, Inc., Hoboken, NJ, 2006.

[28] Krystian Mikolajczyk and Cordelia Schmid. Indexing based on scale invariant interest points. In *IEEE Eighth International Conference on Computer Vision*, volume 1, pages 525–531, July 2001.

[29] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, October 2005.

[30] Adam Milstein, Javier Nicolaṡ Sànchez, and Evan Tang Williamson. Robust global localization using clustered particle filtering. In *Eighteenth national conference on Artificial intelligence*, pages 581–586, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[31] James F. Montgomery, Andrew H. Fagg, and George A. Bekey. The usc afv-i: A behavior-based entry in the 1994 international aerial robotics competition. *IEEE Expert: Intelligent Systems and Their Applications*, 10(2):16–22, 1995.

[32] Hans Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. In *Tech. report CMU-RI-TR-80-03, Robotics Institute, Carnegie Mellon University and doctoral dissertation, Stanford University.* September 1980. Available as Stanford AIM-340, CS-80-813 and republished as a Carnegie Mellon University Robotics Institue Technical Report to increase availability.

[33] Panasonic BB-HCM381 network camera CGI interface technical manual: Web document (referred on 10.3.2007) http://panasonic.co.jp/pcc/products/en/netwkcam/download/us/document/new_cam_cgi_v106a.pdf.

[34] History of mobile robots. http://www.answers.com/topic/mobile-robot.

[35] Andrea Vadaldi's open SIFT implementation. http://vision.ucla.edu/ vedaldi/code/sift/sift.html.

[36] David Lowe's open SIFT implementation. http://www.cs.ubc.ca/ lowe/keypoints.

[37] Rob Hess's open SIFT library. http://web.engr.oregonstate.edu/ hess.

[38] Interest point dectection. http://www.answers.com/topic/interest-point-detection.

[39] Charles A. Rosen and Nils J. Nilsson. Application of intelligent automata to reconnaissance. Technical report, Stanford Research Instite, March, 1967.

[40] Cordelia Schmid, Roger Mohrand, and Christian Bauckhage. Comparing and evaluating interest points. In *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision*, page 230, Washington, DC, USA, 1998. IEEE Computer Society.

[41] S. Se, D. G. Lowe, and J. J. Little. Vision-Based Global Localization and Mapping for Mobile Robots. *IEEE Transactions on Robotics*, 21(3):364–375, 2005.

[42] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. The MIT Press, Massachusetts Institute of Technology, 2004.

[43] R. Sim and G. Dudek. Mobile robot localization from learned landmarks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 1060–1065, Victoria, Canada, October 1998.

[44] Support slides for the book: Introduction to Autonomous Mobile Robots. Web document (Referred on 26.2.2007). http://autonomousmobilerobots.epfl.ch/Slides1.pdf.

[45] Paul Smith, Ian Reid, and Andrew Davison. Real-time monocular SLAM with straight lines. In *Proc. British Machine Vision Conference*, Edinburgh, 2006.

[46] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. The MIT Press, 2005.

[47] Emanuele Trucco and Alessandro Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall, 1998.

[48] Junqiu Wang, R. Cipolla, and Hongbin Zha. Vision-based global localization using a visual vocabulary. In *IEEE International Conference on Robotics and Automation*, pages 4230–4235, Barcelona, Spain, 2005.

[49] Junqiu Wang, Hongbin Zha, and R. Cipolla. Coarse-to-fine vision-based localization by indexing scale-invariant features. *IEEE Transactions on Systems, Man and Cybernetics*, 3(2):413–422, April 2006.

[50] Stanford Research Institute's Artificial Intelligence Center website. http://www.ai.sri.com/shakey.

[51] Jéróme Landré Programming with Intel IPP (Integrated Performance Primitives) and Intel OpenCV (Open Computer Vision) under GNU Linux. Web document (referred on 21.3.2007)
http://downloads.sourceforge.net/opencvlibrary/ippocv.pdf.

APPENDIX A Results for keypoint matching

Image No.	Best five matches and their corresponding nodes				
56	3(N7)	2(N3)	2(N6)	2(N6)	2(N7)
78	4(N3)	4(N5)	4(N6)	4(N7)	4(N7)
98	11(N6)	10(N5)	9(N3)	8(N1)	8(N3)

Table 3: Misclassified images in N1

Image No.	Best five matches and their corresponding nodes				
3	6(N3)	5(N1)	5(N2)	5(N3)	5(N5)
11	8(N3)	8(N4)	8(N6)	7(N2)	7(N4)
14	12(N1)	8(N1)	8(N2)	8(N3)	8(N6)
32	8(N1)	8(N3)	8(N4)	7(N3)	6(N3)
34	7(N1)	7(N3)	6(N3)	6(N4)	6(N5)
46	6(N1)	5(N3)	5(N4)	4(N2)	4(N3)
47	3(N3)	2(N1)	2(N1)	2(N2)	2(N3)
57	6(N1)	6(N2)	6(N3)	6(N3)	6(N4)

Table 4: Misclassified images in N2

Image No.	Best five matches and their corresponding nodes				
4	28(N7)	27(N3)	24(N7)	22(N2)	20(N4)
11	5(N2)	4(N1)	4(N3)	4(N6)	3(N3)
16	6(N1)	6(N1)	6(N3)	6(N3)	6(N4)
20	10(N5)	7(N1)	7(N2)	7(N3)	7(N3)
22	3(N4)	2(N1)	2(N1)	2(N2)	2(N2)
25	30(N7)	29(N3)	28(N7)	26(N4)	25(N5)
34	14(N4)	11(N3)	9(N3)	9(N3)	8(N1)
58	7(N5)	4(N3)	4(N4)	4(N6)	4(N6)
62	9(N4)	7(N1)	6(N3)	6(N7)	5(N1)
66	8(N1)	7(N3)	7(N3)	7(N3)	7(N4)
69	14(N2)	13(N4)	13(N5)	13(N7)	12(N3)
75	7(N1)	7(N3)	6(N3)	5(N1)	5(N5)
83	11(N4)	9(N4)	8(N3)	7(N1)	6(N3)
94	2(N1)	2(N3)	2(N6)	2(N7)	1(N1)
97	28(N2)	27(N7)	23(N3)	20(N6)	19(N7)
103	6(N6)	5(N4)	4(N1)	4(N3)	4(N3)
117	6(N2)	4(N2)	4(N3)	4(N7)	3(N1)
119	6(N4)	5(N1)	5(N3)	5(N4)	5(N4)
130	21(N4)	11(N3)	10(N3)	9(N3)	9(N7)

Table 5: Misclassified images in N3

Image No.	Best five matches and their corresponding nodes				
1	7(N3)	7(N3)	7(N4)	6(N1)	6(N1)
33	6(N2)	4(N2)	4(N2)	4(N3)	4(N3)
70	4(N3)	4(N4)	4(N7)	4(N7)	3(N2)
98	6(N3)	6(N4)	6(N4)	5(N3)	5(N3)

Table 6: Misclassified images in N4

Image No.	Best five matches and their corresponding nodes				
4	1(N7)	-	-	-	-
5	7(N3)	6(N6)	6(N7)	5(N1)	5(N1)
6	6(N4)	4(N3)	4(N4)	3(N1)	3(N3)
18	2(N3)	2(N6)	2(N7)	2(N7)	1(N1)
30	7(N4)	6(N3)	6(N3)	6(N7)	5(N3)
40	6(N1)	5(N3)	5(N4)	4(N2)	4(N3)
55	4(N4)	3(N3)	3(N5)	3(N5)	2(N1)
58	7(N3)	6(N3)	6(N4)	6(N5)	5(N3)
60	7(N4)	6(N3)	6(N5)	6(N6)	5(N2)
61	6(N4)	6(N5)	6(N7)	5(N4)	5(N7)
66	13(N4)	13(N4)	12(N3)	12(N3)	12(N4)
69	16(N3)	15(N4)	13(N1)	13(N3)	13(N3)

Table 7: Misclassified images in N5

Image No.	Best five matches and their corresponding nodes				
12	14(N3)	14(N4)	14(N4)	13(N1)	12(N1)
15	1(N1)	1(N2)	1(N3)	1(N3)	1(N5)
16	4(N1)	4(N3)	4(N3)	3(N1)	3(N3)
17	5(N4)	4(N1)	4(N2)	4(N3)	4(N3)
18	3(N3)	3(N4)	3(N4)	3(N6)	3(N6)
19	4(N5)	4(N6)	3(N1)	3(N3)	3(N6)
24	6(N2)	6(N3)	6(N6)	5(N3)	4(N1)
25	13(N4)	10(N3)	10(N7)	9(N1)	9(N3)
26	6(N2)	5(N6)	4(N1)	4(N2)	4(N6)
31	16(N4)	13(N5)	13(N1)	12(N3)	11(N4)
33	18(N4)	17(N1)	16(N3)	16(N6)	16(N6)
34	10(N4)	8(N6)	8(N7)	7(N2)	7(N6)
41	6(N7)	6(N7)	5(N1)	5(N3)	5(N3)
51	7(N3)	7(N6)	6(N3)	6(N5)	5(N1)
56	7(N4)	7(N6)	6(N5)	5(N4)	5(N6)
61	13(N1)	10(N4)	10(N4)	9(N3)	9(N3)
69	11(N2)	10(N6)	9(N5)	8(N6)	7(N5)
72	33(N1)	24(N3)	24(N3)	23(N1)	23(N4)
74	2(N3)	2(N6)	2(N6)	1(N3)	1(N3)
75	6(N3)	6(N7)	5(N1)	5(N3)	5(N4)
76	8(N3)	8(N4)	6(N4)	6(N6)	5(N1)
77	1(N1)	1(N1)	1(N3)	1(N3)	1(N3)

Table 8: Misclassified images in N6

Image No.	Best five matches and their corresponding nodes				
9	8(N4)	7(N3)	7(N4)	7(N7)	7(N7)
43	10(N5)	10(N7)	8(N7)	7(N3)	7(N6)
46	8(N4)	8(N7)	5(N3)	5(N3)	4(N1)
52	4(N1)	2(N1)	2(N4)	2(N2)	2(N3)
61	7(N3)	6(N1)	6(N5)	6(N4)	6(N7)
66	11(N1)	11(N4)	10(N4)	9(N4)	9(N6)
77	14(N3)	11(N5)	8(N5)	8(N6)	8(N7)
79	6(N3)	6(N4)	6(N7)	5(N3)	5(N3)
80	3(N6)	2(N3)	2(N3)	2(N6)	1(N1)
81	12(N3)	9(N2)	9(N7)	8(N6)	7(N3)

Table 9: Misclassified images in N7

APPENDIX B Examples of training images

Figure 22: Views from N1 - Masters students' office

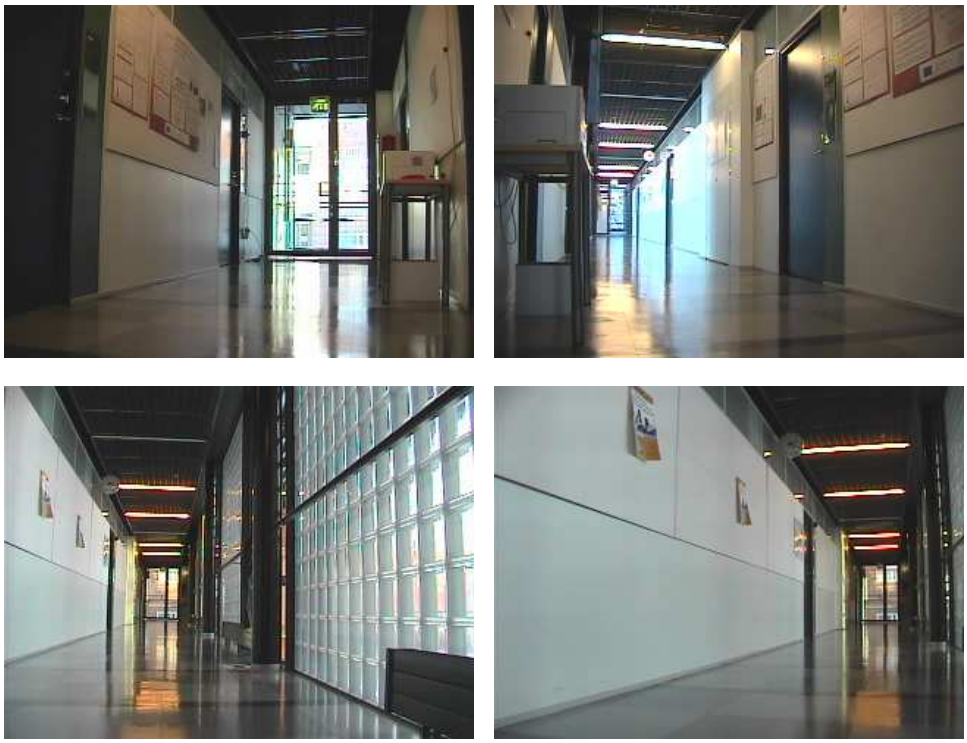


Figure 23: Views from N2 - corridor next to N1

Figure 24: Views from N3 - stairway

Figure 25: Views from N4 - Machine vision Lab

Figure 26: Views from N5 - corridor next to N4

Figure 27: Views from N6 - meeting room

Figure 28: Views from N7 - corridor next to N3