AN IMAGE PROCESSING BASED FAST LOCALIZATION

A DISSERTATION

Submitted in partial fulfillment of the requirements for the award of the degree
of
MASTER OF TECHNOLOGY
in
INFORMATION TECHNOLOGY

By

KUTE MANGESH SHIVAJI

DEPARTMENT OF ELECTRONICS & COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE -247 667 (INDIA)
JUNE, 2007

CANDIDATE'S DECLARATION

I hereby declare that the work, which is being presented in the dissertation entitled "**An Image Processing based Fast Localization**" towards the partial fulfillment of the requirement for the award of the degree of **Master of Technology** in **Information Technology** submitted in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee (India) is an authentic record of my own work carried out during the period from July 2006 to June 2007, under the guidance of **Dr. Ankush Mittal, Associate Professor, Department of Electronics and Computer Engineering, IIT Roorkee.**

I have not submitted the matter embodied in this dissertation for the award of any other degree or diploma.

Date: 08/06/2007

Place: Roorkee

(Kute Mangesh Shivaji)

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

Date: 08/06/2007

Place: Roorkee

(Dr. Ankush Mittal)

Associate Professor

Department of Electronics and Computer Engineering

IIT Roorkee – 247 667

ACKNOWLEDGEMENTS

I would like to extend my heartfelt gratitude to my guide **Dr. Ankush Mittal**, Associate Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, for his able guidance, regular source of encouragement and assistance throughout this dissertation work. It is his vision and insight that inspired me to carry out my dissertation in the upcoming field of Localization using Image Processing. I would state that the dissertation work would not have been in the present shape without his umpteen guidance and I consider myself fortunate to have done my dissertation under him.

I also extend my sincere thanks to **Dr. D. K. Mehra**, Professor and Head of the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee for providing facilities for the work.

I also wish to thank all my friends for their valuable suggestions and timely help.

Finally, I would like to say that I am indebted to my parents for everything that they have given to me. I thank them for the sacrifices they made so that I could grow up in a learning environment. They have always stood by me in everything I have done, providing constant support, encouragement and love.

Kute Mangesh Shivaji

Abstract

The knowledge of the current location is important for navigation purposes. A person generally understands his/her location by using the knowledge of the surrounding landmarks. In the urban environment, buildings can be used to play the role of the landmarks. For this purpose the building has to be reliably and efficiently recognized by matching its images with its image model.

In this work we propose a system for fast localization using mobile phone camera imagery. It is a hierarchical approach for localization. It involves two stages. In first stage, it identifies building in the query view using wide baseline matching algorithm. Our wide baseline matching algorithm is itself a three step approach for recognition of buildings. In the first step, we use previously proposed global representation named as colour histogram. Using this representation a small number of candidate images are selected from the database. In the second step, we use shape matching technique. Using shape context candidate images from previous step are further filtered out. In the third step, images are compared by using their local feature points. We propose a novel algorithm for locating feature points of the image and a clustering model to match different regions of the image. This clustering avoids random matches and makes the matching procedure independent of number of feature points of the image. In second stage, the system identifies view point of query image. We propose novel viewpoint identification algorithm for this purpose. This system is validated by experiments carried out with database consisting of images acquired by us in different light and weather conditions by using cell phone camera and also with publicly available database ZuBuD.

Table of Contents

List of Figures

1 Introduction

1.1 Overview

In mobile computing, location based information services allow many promising applications. One of these applications is to provide a navigation support system to travelers. This report presents our work on fast localization of a user carrying a mobile camera in an urban area based on images captured using the camera. Such image based localization technology is particularly useful in unfamiliar high-rise urban areas, where GPS triangulation may not be adequately achieved due to line-of sight obstruction.

With the wide dissemination of digital cameras, images acquired by the cameras can provide means for determining the position of a person in the urban area. Such a system can be achieved as a two stage process: by first acquiring a database of buildings and/or locations of particular area from different viewpoints, followed by recognition of a new query view by matching it to the closest model in the database. The visual navigation system seems like a task of matching a query image to an image in the database to determine the location information. Considering the developments in content based retrieval techniques, it may at first seem to be a trivial task. However, the wide disparity in the imaging conditions of the query images and the database images make the task difficult. There are several other challenges for developing a system with recognition accuracy suitable for real time deployment. Some of the challenges are as follows

1. Query image may be at a different scale.

   Users may send an image to the system representing objects at different scales than the one present in the database. It is not possible to store images at all possible scales of each location. So the system must be able to deal with variation in the scale.

2. Query image may be taken from different viewpoints.

   It is not practically feasible to store images of each location from every possible view point. So the system must be able to recognize similar images differing in the viewpoint.

3. Occlusions

   Other objects are likely to be occluded the location bject of interest. The images stored in the database can be acquired under i conditions. But when the user captures an image, there are likely to be ot objects occluding the location or object of interest. The recognition algorithn eds to be robust to the presence of occlusion.

4. Variation in illumination

   The query image provided by the user would be taken a ferent time of the day. Also during different weather conditions, light con ons are different. So illumination of the query image and database image are ely to be different. So, the recognition algorithm needs to be robust to variatior mination.

5. Operation speed

   The objective is to provide interactive system to tl iser to locate him in unknown territory. So the response time of the system is most important factor for the success of the system.

6. Heterogeneity

   As images are captured from diverse sources, image par ters such as resolution and colour depth are likely to vary.

1.2 Problem Statement

The goal of this work is to develop the localization system to l e user in the unknown territory using mobile phone imagery.

1.3 Organization of Report

This work proposes the hierarchical system for urban navig n. In the first step it recognizes the building in the query view and in second step it s out view point of the query view. The database for a wide baseline matching algorith onsists of few (3 to 5) views of buildings in the area. Some of related work is discu l in section 2. For the purpose of location recognition and wide-baseline matching e propose the novel approach to locate feature points of the image. These feature p s are used to represent the image. This approach is described in section 3.3. Given a new query view. the location recognition phase is accomplished by a voting scheme. In section 3.4, we propose the novel approach to determine the view point of the image. It uses Harris corner detector [1] to locate feature points of the image. After recognizing building by the wide baseline matching algorithm. a view point determination algorithm determines view point of the query image. Implementation issues are discussed in section 4. Section 5 gives some experimental results and highlights the accuracy and efficiency of the system.

2 Literature Review

2.1 Related Work

The general approach is a wide baseline matching, in order ind the images in the database. Being able to match under conditions mentioned in ion 1, it allows one to keep the number of reference images in the scene model to minimum. Most wide baseline matching approaches are based on so-called inva nt regions. These are constructed around interest points, such as corners, in a way a adapt their shapes to the viewpoint and keep the part of the scene they enclose fixed.

These regions are then described by a descriptor vector, th lements of which are invariant under combinations of geometric and photometri ianges. They can be matched efficiently between views taken from different view ts and under different illumination. The crux of the matter is that these affine invaria egions are determined solely on the basis of a single image, i.e. no information a t the other view(s) is necessary during extraction.

The problem of localization has been addressed by several auth [2-9]. The differences between approaches lie in the way in which interest points, al image regions and descriptor vectors are extracted. All these approaches work wel th high quality images taken with digital camera. But these approaches do not work for v quality images taken with mobile phone camera. Images of average mobile phone c ra are blurred, having very low contrast. Colour information is not so faithful.

In [2] authors used vanishing direction for alignment of a bu ng view in the query image to the canonical view in the database and proposed m ing using descriptors associated with interest regions, followed by the relative po ecovery between the views from planar homographies. However, conducting two vic matching between the query view and every nearby database view is slow. And also t system can not able to distinguish between similar buildings, without using more information, e.g. extra query views.

The approach in [3] works by computing affinely invariant Fourier features from intensity profiles in each image. It uses Harris corner detector for extracting interest point. All possible pairs of interest points are formed. Pairs consisting of points which are very close or far away are rejected. For each pair of interest points, the image intensity profile for line going between two points is extracted. Each intensity profile is described by vector of Fourier coefficients. For each feature vector in the query image, all similar feature vectors in reference image are found. Using a voting scheme, pairs of corresponding interest points are found. However, this approach is not intensity invariant. It does not find pose of the query view.

In [4], a new set of image elements that are put into correspondence, the so called extremal regions, are introduced. Extremal regions possess highly desirable properties: the set is closed under continuous (and thus projective) transformation of image coordinates and monotonic transformation of image intensities. An efficient (near linear complexity) and practically fast detection algorithm (near frame rate) is presented for an affinely-invariant stable subset of extremal regions, the maximally stable extremal regions (MSER). The MSER is represented by position of a local intensity minimum and a threshold. A new robust similarity measure for establishing tentative correspondences is proposed. However this approach does not work properly for low contrast images.
Given a photograph of a building, the system described by Shao [5] can identify more photographs of the same building in a large database of photographs obtained from a wide range of viewpoints. However their system also does not determine the pose of the query. Coorg and Teller [6] describe a system that uses vanishing points to determine the orientation of dominant building facade relative to panoramic camera. However a combination of specialist hardware is used to determine camera position, including GPS and inertial sensors.

In [10], authors suggested a two stage hierarchical approach fo :alization. In first stag, it uses global features, named as colour histogram to select a : ll number of candidate images for match. In the second stage, images are compar by using SIFT (Scale Invariant Feature Transform) [11, 20] feature points. Severa her authors also used SIFT to locate feature points. As SIFT considers local extrema :r laplacian pyramid as a feature point, it does not able to deal with occlusions. Eith ne to one matching or clustering in descriptor space is used by these authors. All th techniques may cause random matching, as image may have same local conditions in :rent regions.

One of the central issues pertinent to the recognition problen the choice of suitable representation of the class and its scalability to large number o: :mplars. In the context of object recognition, both global and local image descripte ıave been considered. From the perspective of the application the efficiency of approach has to be considered. The methods which employ solely geometric l local feature based matching techniques are often quite slow as explained in [12] ıerefore, when dealing with large databases, it's desirable to have some simple indexir :ctor for all models, so that unlikely models can be eliminated in advance. Global desc ›rs can provide a good solution.

2.2 Research Gaps

From study of related work, we can conclude that

1. Existing image matching algorithms are less efficient fo r purpose.
2. They require large number of images in the database.
3. Their accuracy decrease for low quality images like mol images.
4. They are not scalable.
5. Commonly used algorithm SIFT can not able to deal wit :clusions.
6. Commonly used one to one matching technique for des .or matching may lead to random matches.
7. Available view angle determination algorithms requir ldditional hardware or additional information to determine view angle.

3 Location Recognition System

3.1 System Overview

The prior data available is a 3D geometric model layout of the urban area. The framework involves separate stages of recognizing buildings, followed by pose computation. The architecture of the system is shown in figure 3.1.

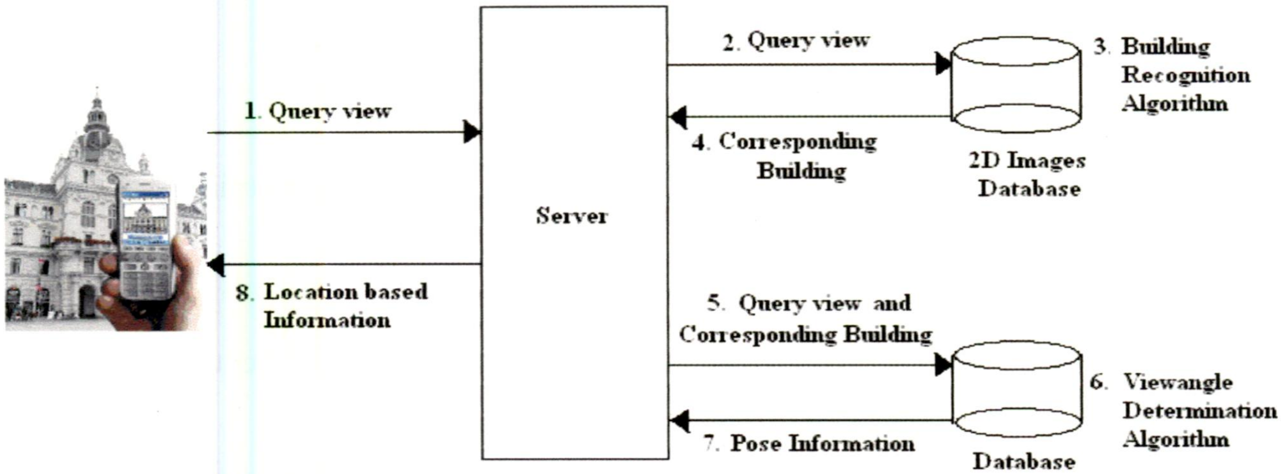


Figure 3.1 Architecture of the system.

Our work involves two stages. First one is recognition of building by applying image matching algorithm. The database for this algorithm consists of few views of the buildings in the area under consideration. This stage is a hierarchical approach. In first step it uses global feature of the image for locating possible matches of the query image. It reduces search area considerably. In second step it uses dominant shapes of the image to further filter out probable candidate images. In third step local features of the image are used for finding exact match for the query image. Second stage is for finding view point of the query image. It takes building identified in first stage as input and determines view point from which the query image has snapped. It uses database images of building taken at angles separated by 15 degree.

3.2 Image Features

Our aim is to match the query image with database images an cognize the building in the query image. For matching purpose, we need a way to repr t each image uniquely. As we discussed earlier, there are possibilities of occlusior ariations in intensity, changes in scale, changes in view angle. So it needs a chanism to construct representation of the image which is invariant in those cases.

Generally there are two ways to describe image. They are bal features and local features.

3.2.1 Global Features

Global features consider entire image. It provides representatio r entire image. Mainly there are three global features, which are used widely. They are

1. Colour Histogram

   It describes colour distribution of the image. It generate e vector per image.
2. Texture

   Image texture is defined as a function of the spatial v ion in pixel intensities (gray values). One immediate application of image te is the recognition of image regions using texture properties. It is useful for ours that are sitting on the surface, rather than running through the material pictures on the wall, printed/painted logos, text, etc.
3. Contours

   Pixels in an image which have some similar feature. ther this is a specific intensity or a texturing pattern, can be grouped together contour is then defined as an outside boundary of such group. Contour lines pr e global representation of the image.

Global features provide compact representation of an image. t they are sensitive to occlusion and clutter. They require segmentation.

3.2.2 Local Features

Local features describe localized image regions. Descriptors are computed around interest points. There is no need for segmentation. Local Features are robust to occlusion and clutter. But local features represent images using different size sets of feature vectors. Local features do not lend themselves easily to standard classification techniques.

3.3 Building Recognition

After considering both global and local features characteristics, we decided to use both features in combination to exploit their advantages. To make use of both global and local features, we propose a hierarchical approach for building recognition. It is a three step approach. In the first step, it uses global characteristics of the image, in second step it uses dominant shapes of the image to further filter out probable candidate images and in third step it uses local scale, rotation invariant features of the image.

In the first step, a colour histogram, a global descriptor of image as suggested in [10], is used. Using this descriptor, a few closest images are selected. These closest images are used as probable candidate images in a second step. As each building has characteristics shapes, second step uses those shapes to further filter out candidate images. The third step uses local features of an image for selecting exact match for a query image. In third step we propose novel approach to locate scale, rotation, intensity invariant feature points of an image. The entire approach is explained in detail in following subsections.

3.3.1 Identifying Candidate Images using Global Descriptor

This is a first step of building recognition algorithm. As proposed in [10], colour histogram is used as global descriptor of the image. Realizing the task of identifying candidate images utilizes the fact that buildings contain constrained geometric structures, such as planar structures. The orientation of most of the pixels complies with the edges of the building. The direction, closer to vertical direction, along which most of the pixels have their orientation, is detected as dominant vertical direction and the direction, closer to horizontal direction, along which most of the pixels have their orientation, is detected as dominant horizontal direction [13]. The colour distribution y based on the pixel whose orientation complies with dominating directions is nputed. The colour histogram constructed considers only hue of pixels, to nullif e effect of intensity variation while taking photos.

3.3.1.1 Colour Histogram

This subsection describes the procedure to construct colour his am. The edges from the image are located using the canny edge detector [14]. The xels along the edges detected are grouped together using the connected component rithm. In connected component algorithm, orientation of pixels is used for assi g label to pixel. If difference in orientation of adjacent pixels is below threshol hen both pixels are assigned to same group. Otherwise both pixels are assigned eparate groups. The dominant components, the component having length greater th % of the image size (maximum of the two sides of the image), are located.

The pixels of the dominant components are grouped according r orientation. Twelve bins are used for grouping. Then the peaks are located in gram. Peaks in the histogram indicate dominant directions in the image. The dom t direction closest to vertical direction is considered as dominant vertical direction an e dominant direction closest to horizontal direction is considered as dominant horizont irection.

The hue of pixels of the image having orientation in dominan rtical and horizontal direction is calculated using the formula.

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.2125 & 0.7154 & 0.0721 \\ -0.1150 & -0.3850 & 0.5000 \\ 0.5000 & -0.4540 & -0.0460 \end{bmatrix} \quad (1)$$

$$H = \frac{\tan^{-1}(C_b, C_r)}{\pi}, \quad -1 \le H \le 1 \quad (2)$$

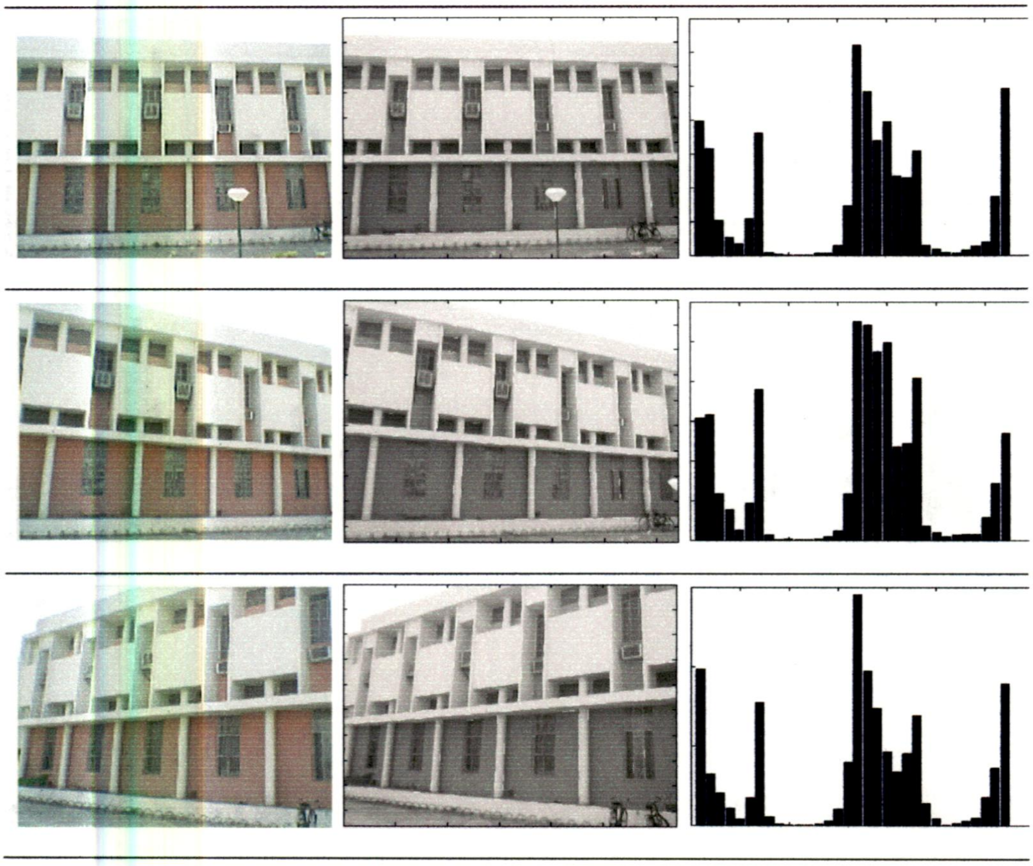


Figure 3.2 Similar colour histogram extracted from three different views of the same building. First column shows original images. Middle column shows the dominant components in the images. Third column shows the Colour histograms of the images.

The hue histogram for each dominant direction is constructed. Sixteen bins are used for constructing the histogram for each dominant direction. The hue histograms of dominant directions are combined (histogram of vertical dominant direction followed by histogram of horizontal dominant direction) to form the colour descriptor of the image. Thus, it is a 16 x 2 = 32 dimensional vector. The colour histogram of the image is normalized. The colour histograms of the three views of the same building are shown in the figure 3.2. The colour histograms of the three different building images are shown in the figure 3.3.

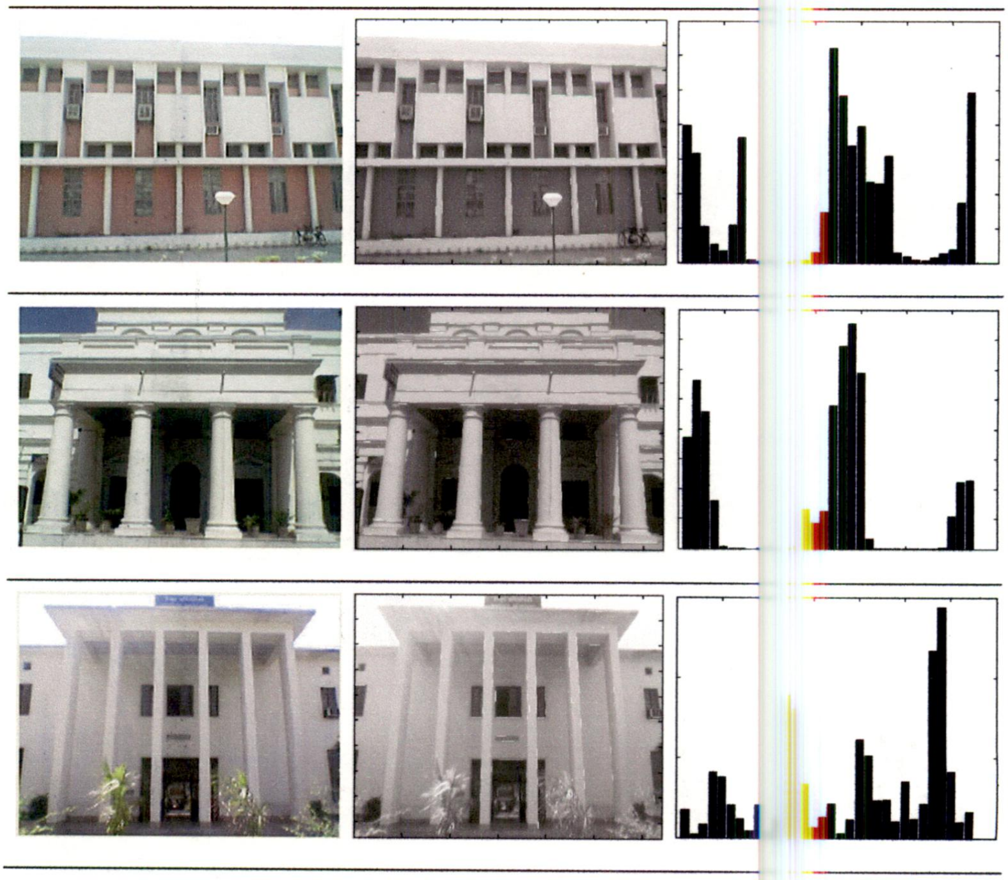


Figure 3.3 Different colour histogram extracted from three different building images. First column shows original images. Middle column shows the dominant components in the images. Third column shows the Colour histograms of the images.

3.3.1.2 Retrieval of Candidate Images

This subsection describes the matching process. The colour histogram of the query image is compared with colour histograms of database images. To compare the query image with the database image X^2 distance is used. Given colour histogram vectors of an input image h_t and database image h_p, the X^2 distance is defined as,

$$X^2(h_t, h_p) = \sum_k \frac{(h_t(k) - h_p(k))^2}{h_t(k) + h_p(k)} \quad (3)$$

where k is number of histogram bins.

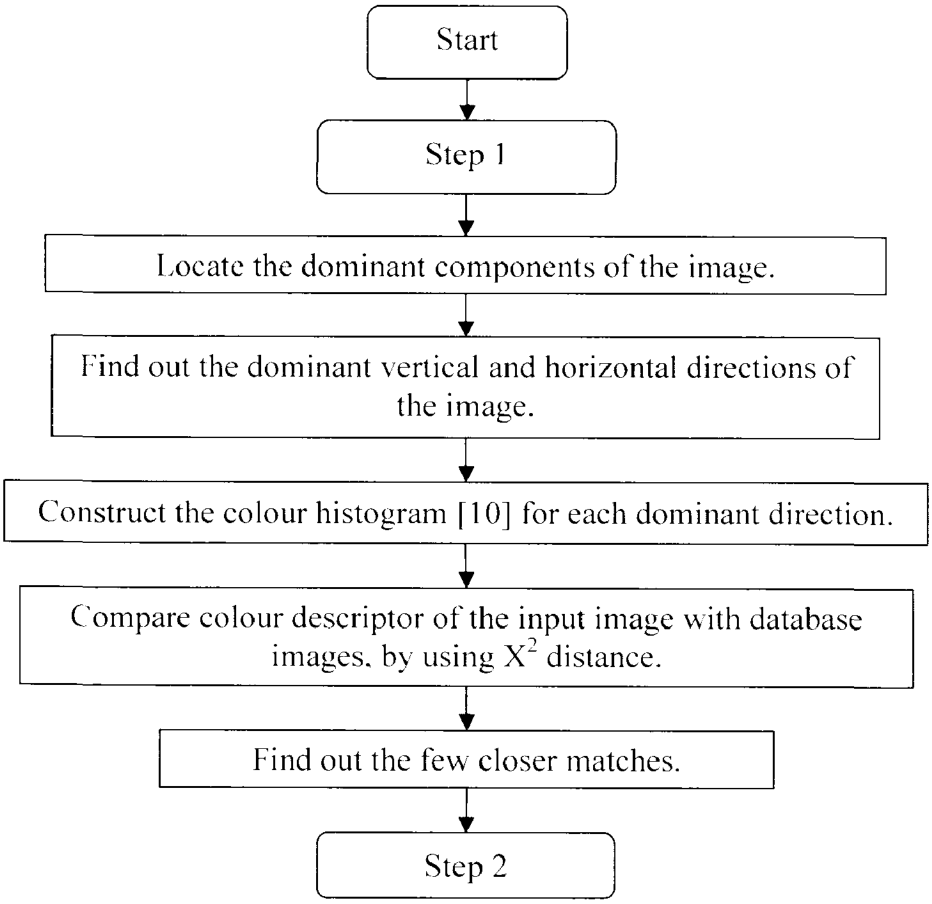


Figure 3.4 Block diagram describing steps in identifying candidate images using global descriptor.

The cardinality of the output subset is calculated by considering the ambiguity of the recognition. The formula used is

$$C_p = n \times \frac{X_1^2}{\frac{1}{(n-1)} \sum_i X_i^2} \qquad (4)$$

where n is the maximum number of images in the output set. X_1 is lowest value. We take n = 30. $i \in \{2,.....,n\}$. The output subset is used in next stage to find out the closest match. The block diagram of the step 1 is given in figure 3.4. Example of step 1 output is given in figure 3.5.

Query Image 1

Output of stage 1 step1

1 2 3 4

Figure 3.5 Example of step 1 output using database 1 (described in section 5).

3.3.2 Identifying Candidate Images using Shapes

This subsection describes second step of building recognition algorithm. After identifying candidate images using colour histogram, we use shapes of building to further filter out probable candidates. The task of identifying probable candidate images utilizes the fact that each building has certain geometric structures, such as tomb, arrangements of windows etc. Such geometric structures can be used to describe building structure. Shapes of building are identified and described by using shape context. These descriptors are compared together to identify probable candidate images.

3.3.2.1 Shapes of the image

Shape context matching [25] can be understood as a point set matching technique. It is invariant to scale and translation. It is roboust to large extent to rotation and deformation. For shape context analysis the edge elements of shape are considered as feature points. They need not correspond to key points such as maxima of curvature. The concept of shape context is used to describe each dominant shape of the image.

For locating shapes of building, edges from the image are located using the canny edge detector [14, 28]. Then the connected component algorithm is used for grouping pixels along the edges detected. Orientations of pixels are used for grouping pixels. If difference in orientation of adjacent pixels is below threshold, then both pixels are assigned to same group. Otherwise both pixels are assigned to separate groups. The component having length greater than 5% of the image size (maximum of the two sides of the image), are identified as dominant components of the image. Centers of dominant components are used for constructing shape context.

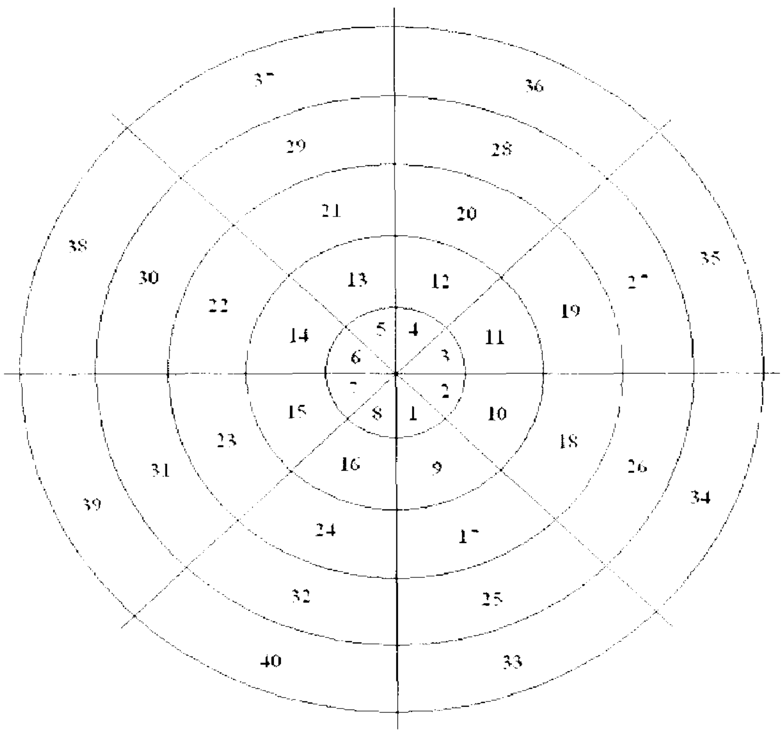


Figure 3.6 Calculation of Shape Context descriptor.

Shapes of the image are described by using descriptor known as shape context [26, 27]. Shape context is a 3D histogram of edge point locations and orientations. To compute descriptor a polar location grid with five bins in radial direction (the radius set to 5, 10, 15, 20 and 25) and eight bins in angular direction around feature point is considered. Total forty bins are considered. A point contribution to the histogram is weighted with the gradient magnitude. It results into 5 x 8 matrix. This matrix i ad row major wise to construct 40 dimensional vector as descriptor.

3.3.2.2 Retrieval of Candidate Images

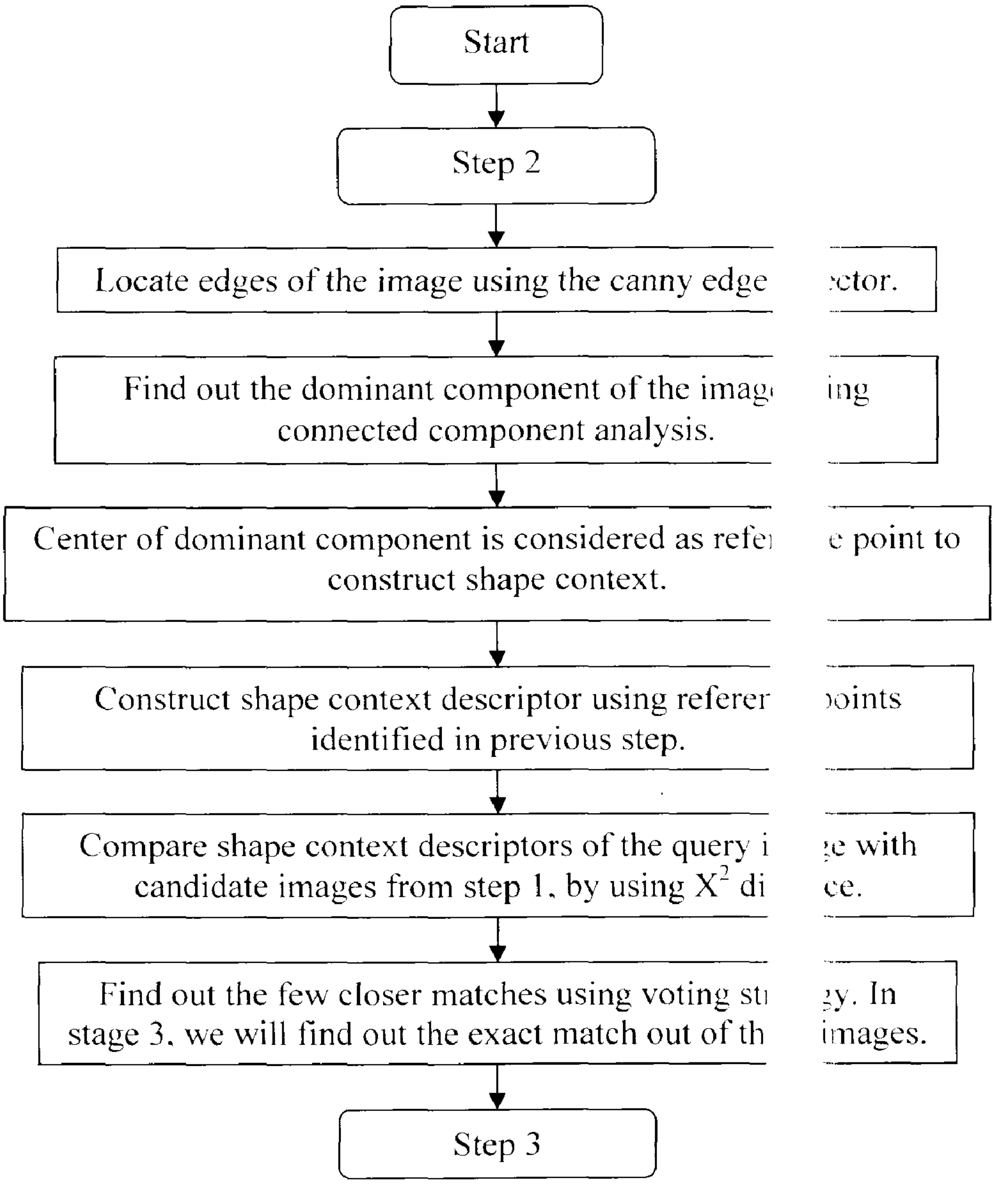


Figure 3.7 Block diagram describing steps in identifying candi images using shapes.

The shape context descriptors of the query image are com d with shape context descriptors of database images selected in step 1. To co re the shape context descriptor X^2 distance (equation 3) is used. The cardinality of the output subset is calculated by considering the ambiguity of the recognition. The formula used is

$$C_p = n \times \frac{X_1^2}{\frac{1}{(n-1)}\sum_i X_i^2} \tag{5}$$

where n is the maximum number of images in the output set. We take n = no of images selected in step 1. $i \in \{2, \ldots, n\}$. The output subset is used in next step to find out the closest match. The block diagram of the step 2 is given in figure 3.7.

3.3.3 Exact Match for Query Image

This subsection describes third step of building recognition algorithm. The purpose of the third step is to refine the results of the second step and find the exact match. In this step the local feature points and their descriptor are used to match the images. In this section, we propose the novel approach for locating scale, intensity and rotation invariant feature points of the image.

3.3.3.1 Feature Point Selection

We need a mechanism to locate feature points of an image. Two important characteristics of the feature point are distinctiveness and repeatability. A feature which appears in multiple views of the same model is more repeatable and likely to appear in a new view. On the other hand, a feature which appears in multiple models is less distinctive than those present only in views of single model. The repeatability and distinctiveness of each feature can be characterized by the Parzen window method [15]. The parzen probability would be higher when the feature is more repeatable and characteristics and low otherwise. For each image we consider only those features with probability higher than certain threshold.

3.3.3.2 Scale space theory

As the scale of the query image may be different from the scale ε hich database images are snapped. Therefore our mechanism to locate feature point s ld be scale invariant. So we need a mechanism to represent image at different sca 'ollowing subsection discuss about need of such mechanism and a way to represent it.

3.3.3.2.1 Need of scale space theory

An inherent property of real-world objects is that they only ex s meaningful entities over certain ranges of scale. A simple example is the concept of 'anch of a tree, which makes sense only at a scale from, say, a few centimeters to at st a few meters; it is meaningless to discuss the tree concept at the nanometer or l neter level. At those scales, it is more relevant to talk about the molecules that form ı eaves of the tree, and the forest in which the tree grows, respectively. This fact, that ol ts in the world appear in different ways depending on the scale of observation, has imp nt implications if one aims at describing them. It shows that the notion of scale is of ut .t importance.

To be able to extract any information from image data, one obvi ly has to interact with it using certain operators. The type of information that caı : obtained is largely determined by the relationship between the size of the actual strı res in the data and the size (resolution) of the operators (probes). Some of the very lamental problems in image processing concern what operators to use, where to apply m and how large they should be. If these problems are not appropriately addressed, th ıe task of interpreting the operator response can be very hard.

In certain circumstances, it may not be obvious at all to deterı : in advance what are the proper scales? Besides the inherent multi-scale properti ›f real-world objects (which, in general, are unknown), such a system has to fa‹ he problems that the perspective mapping gives rise to size variations, that noise is roduced in the image formation process, and that the available data are two-dimens al data sets reflecting indirect properties of a three-dimensional world. To be able to ‹ : with these problems, an essential tool is a formal theory for how to describe image structures at different scales.

3.3.3.3 Scale-space Representation

Scale-space theory [12, 21, 22 and 24] is a framework for visual operations to handle the multi-scale nature of image data. A main argument behind its construction is that if no prior information is available about what are the appropriate scales for a given data set, then the only reasonable approach for an uncommitted vision system is to represent the input data at multiple scales. This means that the original signal should be embedded into a one-parameter family of derived signals, in which fine-scale structures are successively suppressed.

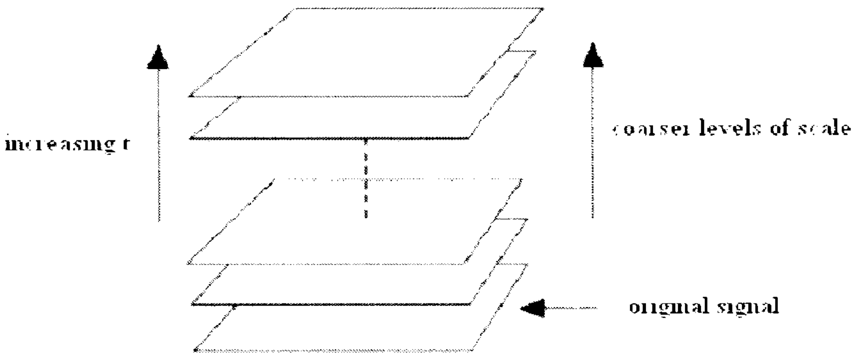


Figure 3.8 A multi-scale representation of a signal is an ordered set of derived signals intended to represent the original signal at different levels of scale.

A crucial requirement for this is that structures at coarse scales in the multi-scale representation should constitute simplifications of corresponding structures at finer scales. They should not be accidental phenomena created by the method for suppressing fine-scale structures. This idea has been formalized in a variety of ways by different authors. The convolution by the Gaussian kernel and its derivatives is singled out as a canonical class of smoothing transformations. The requirements (scale-space axioms) that specify the uniqueness are essentially linearity and spatial shift invariance, combined with different ways of formalizing the notion that new structure:)uld not be created in the transformation from fine to coarse scales.

In summary, for any N-dimensional signal $f : R^N \rightarrow R$, its scal ace representation L: $R^N \times R_+ \rightarrow R$ is defined by

$$L(x;t) = \int_{\xi \in R^N} f(x-\xi)g(\xi)d\xi \qquad (6)$$

where $g : R^N \times R^+ \rightarrow R$ denotes the Gaussian kernel

$$g(x;t) = \frac{1}{\left(2\pi\sigma^2\right)^{N/2}} e^{-(x_1^2+\ldots+x_N^2)/2t} \qquad (7)$$

and the variance t of this kernel is referred to as the scale param

Equivalently, the scale-space family can be obtained as the ution to the (linear) diffusion equation

$$\partial_t L = \frac{1}{2}\nabla^2 L \qquad (8)$$

with initial condition L(. ; t) = f. Then, based on this re sentation, scale-space derivatives at any scale t are defined by

$$L_{x^\alpha}(.;t) = \partial_{x_1^{\alpha_1} \ldots x_N^{\alpha_N}} L(.;t) = (\partial_{x_1^{\alpha_1} \ldots x_N^{\alpha_N}} g \quad) * f \qquad (9)$$

3.3.3.4 Feature point localization

Now our aim is to locate scale, rotation, intensity invariant fe :s points of an image. Commonly used method to locate feature points is Scale Inv nt Feature Transform (SIFT). It is an approach for detecting and extracting local f e descriptors that are reasonably invariant to changes in illumination, image noise, r on, scaling, and small changes in viewpoint. Interest points for SIFT features corre d to local extrema of difference-of-Gaussian filters at different scales. The first ste ward the detection of interest points is the convolution of the image with Gaussian ers at different scales, and the generation of difference-of-Gaussian images from t lifference of adjacent blurred images. Interest points (called keypoints in the SIFT framework) are identified as local maxima or minima of the DoG images across scales. If the pixel is a local maximum or minimum, it is selected as a candidate keypoint. Major flaw in this method is that it can not able to deal with occlusions like tree, human being. So we need a new method to locate feature points of an image.

Here we introduce a novel approach to locate feature points of an image. To identify scale invariant features of the image, the scale space of the image is used. We wish to identify locations in image scale space that are invariant with respect to image translation, scaling and rotation and are minimally affected by noise and small distortions.

Lindeberg [12] has shown that under some rather general assumptions on scale invariance, the Gaussian kernel and its derivatives are the only possible smoothing kernel for scale space analysis. To achieve a high level of efficiency, we have chosen to select key locations over a difference of Gaussian function applied in scale space. This can be computed very efficiently by building an image pyramid.

The scale space of an image is defined as a function $L(x, y, \sigma)$ that is produced from the convolution of a variable scale Gaussian $G(x, y, \sigma)$ with an input image $I(x, y)$.

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \quad (10)$$

where $*$ is the convolution operation in x and y, and

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \quad (11)$$

As the 2D Gaussian function is separable, its convolution with the input image can be efficiently computed by applying two passes of the 1D Gaussian function in the horizontal and vertical directions.

Original Image

Feature points detected using standard SIFT algorithm

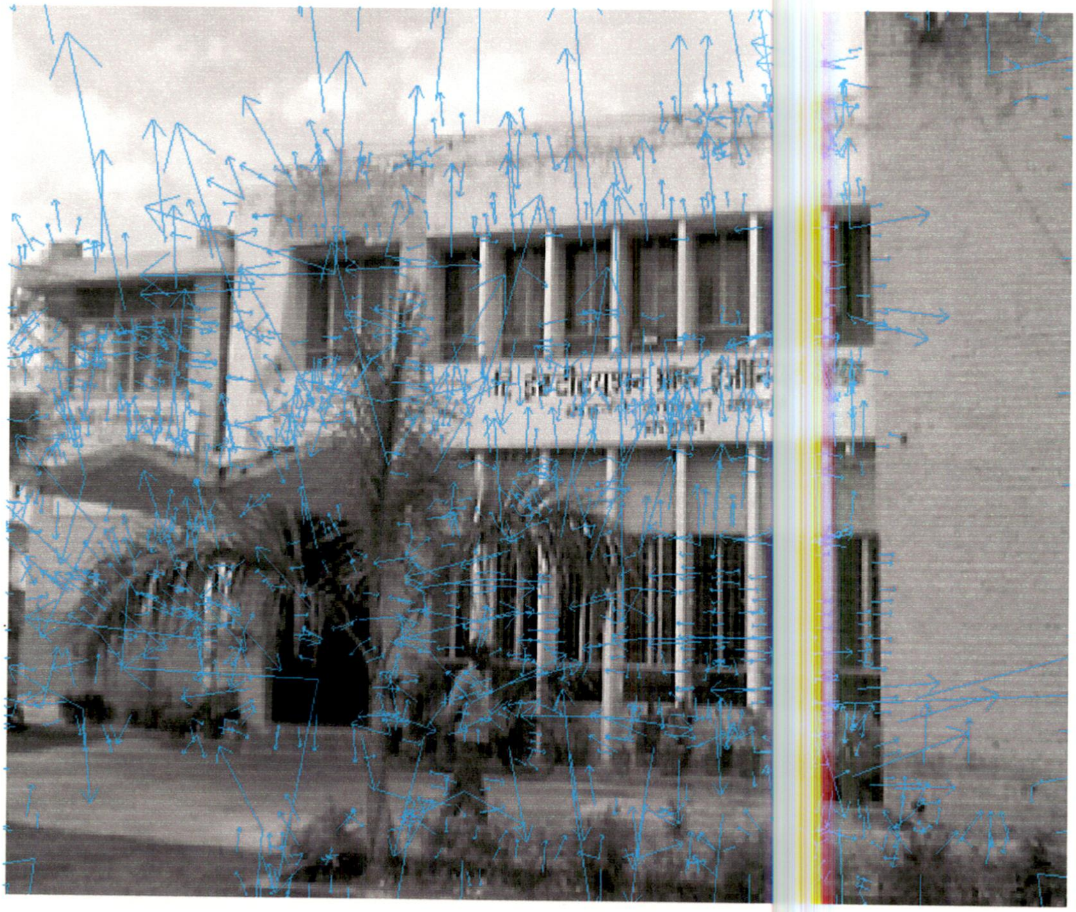


Figure 3.9 Feature points extracted using Standard SIFT algorithm. Standard SIFT algorithm can not deal with occlusions and clutters. Software program provided by author at 'http://www.cs.ubc.ca/~lowe/keypoints/' is used .

Feature points detected using our algorithm

Figure 3.10 Feature points extracted using our algorithm. It locates feature points only from building image and remove occlusions and clutters.

For key localization, all smoothing operations are done using $\sigma = 1.2$, which can be approximated with sufficient accuracy using a 1D kernel with 7 sample points. The input image is first convolved with the Gaussian function using $\sigma = 1.2$ to give an image A. This is then repeated a second time with a further incremental smoothing of $\sigma = 1.2$ to give a new image, B, which now has an effective smoothing of $\sigma = 1.44$. The difference of Gaussian function is obtained by subtracting image B from A, resulting in a ratio of $1.44/1.2 = 1.2$ between the two Gaussians. We chose $\sigma = 1.2$ by doing trade off between processing time and number of feature points to represent image. To generate the next pyramid level, the scale factor is squared.

Building images contain constrained geometric structures, s as planar structures. Edges of the building provide information about geometri ructure. Edges of the building appear in all view of the building. The gradient distri ons, around the points are intensity invariant. So the gradient distribution, around the nts along the edges of the building, can be considered as features of the image. Distin eness and repeatability of these features can be proved by the Parzen window method. considering orientation of a feature point, we can make the descriptor of feature point r on invariant.

After constructing difference of Gaussian pyramid, the canny e detector [14, 23 and 29] is used for detecting edges of the building. The effici connected component algorithm is applied for locating dominant components, the nponent having length greater than certain threshold. Then pixels along the edges det l are grouped together using the connected component algorithm. In connecte omponent algorithm, orientation of pixels is used for assigning label to pixel. If d ence in orientation of adjacent pixels is below threshold, then both pixels are gned to same group. Otherwise both pixels are assigned to separate groups. We gradient direction for connected component algorithm. This allows removing occlusi from the image, as for most of the possible occlusions like tree, human being grad orientations of edge pixels are scattered. The selected components are then sample locate features of the image.

3.3.3.5 Feature point descriptors

This subsection describes procedure to construct descriptor f ature points. Given a stable location, scale and orientation for each key, it is now pos e to describe the local image region in a manner invariant to these transformations. (approach to this is to use gradient magnitude and orientation of pixels in the neig urhood of the feature point [16]. We use gradient location and orientation histogram OH) [16] to describe local image region around feature points.

GLOH is an extension of the SIFT descriptor designed to increase its roboustness and distinctiveness. To compute descriptor a polar location grid with three bins in radial direction (the radius set to 6, 11 and 15) and eight bins in angular direction around feature point is considered. As central bin is not divided into angular direction, total seventeen bins are considered. For each bin, the gradient orientation histogram, using sixteen bins, weighted by gradient magnitude is constructed. It gives 272 dimensional vector as a descriptor of the feature point. In the figure 3.11 a polar location grid is shown.

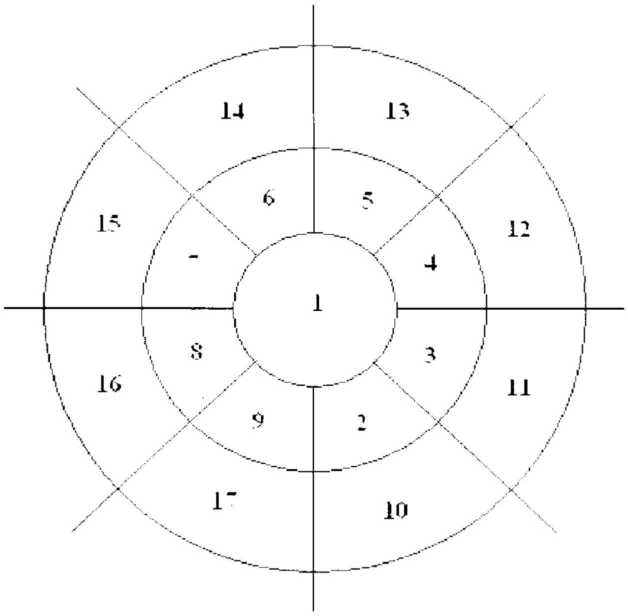


Figure 3.11 Calculation of feature point descriptor (GLOH).

3.3.3.6 Matching

This subsection describes procedure of matching feature point descriptors of query image with that of database images. For indexing, we need to store descriptors of feature points for database images and then identify matching descriptors from query image. The descriptors of the feature points of the query image are compared with the descriptors of feature points of the images selected in step 2. Descriptors of two features are considered as matched when the cosine of angle between their descriptor is above some threshold T_c.

$$\cos(\angle f_a, f_b) = \frac{f_a^T f_b}{\|f_a\|_2 \|f_b\|_2} \qquad (12)$$

The problem with this method is possibility of random m ing. As feature point descriptors represent local region of the image, it does contain any global information. There is always a possibility that local regions o : image, like window, are repeated. Therefore local region of the database image may tch with some random region of the query image. It may lead to false recognition. To : d this problem, we use clustering of feature points in image space.

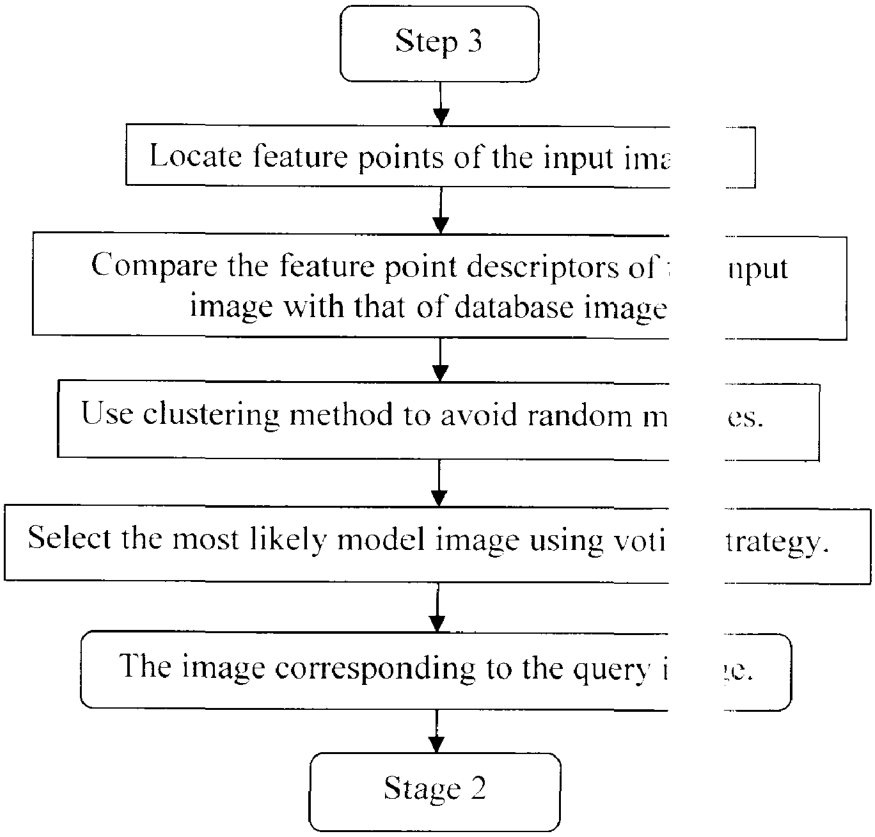


Figure 3.12 Block diagram describing steps in locati :xact match.

First we match feature points of the query image with the database image under consideration as described above. Feature points of the image are grouped together according to their parent component. For each group of the query image, group of the database image under consideration, with which above threshold number of feature points is matched, is located and that is considered as a match for group of query image. If there is no database image group having number of matches above threshold, then query image group is considered as unmatched, even though some of its points have matches in database image.

Voting strategy is used to select the most likely image i.e. image with the largest number of successfully matched points is selected as the best match for the query image. The block diagram of the step 3 is given in figure 3.12.

3.4 Finding Viewpoint of Query Image

After recognizing building in query mage in first stage, the purpose of the second stage is to find the view point of the query image. The geometric features of the building are used for this purpose. In this section, we propose the novel approach for finding out view point of the image. The basis of this algorithm is the fact that relative location of certain key points, like corners in case of building, with respect to center, changes with change in view angle. Harris corner detector can be used to identify required key points of the image. It is discussed in next subsection.

3.4.1 The Harris corner detector

The Harris corner detector [1, 17] is a popular interest point detector due to its strong invariance to rotation, scale, illumination variation and image noise. The Harris corner detector is based on the local auto-correlation function of a signal; where the local auto-correlation function measures the local changes of the signal with patches shifted by a small amount in different directions.

Given a shift (Δx, Δy) and a point (x, y), the auto-correlation function is defined as.

$$c(x,y)=\sum_{w}\left[I(x_i,y_i)-I(x_i+\Delta x,y_i+\Delta y\right. \tag{13}$$

where I(·, ·) denotes the image function and (x_i, y_i) are the nts in the window W (Gaussian) centered on (x, y). The shifted image is approxima by a Taylor expansion truncated to the first order terms,

$$I(x_i+\Delta x,y_i+\Delta y)\approx I(x_i,y_i)+\left[I_x(x_i,y_i)I_y(x_i,y_i)\right]\begin{bmatrix}\Delta\\ \Delta\end{bmatrix} \tag{14}$$

where I_x(·, ·) and I_y(·, ·) denote the partial derivatives i and y, respectively. Substituting approximation equation (14) into equation (13) yie

$$c(x,y)=\sum_{w}\left[I(x_i,y_i)-I(x_i+\Delta x,y_i+\Delta y)\right.$$

$$=\sum_{w}\left(I(x_i,y_i)-I(x_i,y_i)-\left[I_x(x_i,y_i \quad x_i,y_i)\right]\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix}\right)^2$$

$$=\sum_{w}\left(-\left[I_x(x_i,y_i)I_y(x_i,y_i)\right]\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix}\right)^2$$

$$=\sum_{w}\left(\left[I_x(x_i,y_i)I_y(x_i,y_i)\right]\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix}\right.$$

$$=\begin{bmatrix}\Delta x & \Delta y\end{bmatrix}\begin{bmatrix}\sum_{w}(I_x(x_i,y_i))^2 & (x_i,y_i)I_y(x_i,y_i)\\ \sum_{w}I_x(x_i,y_i)I_y(x_i,y_i) & \sum_{w}(I_y(x_i,y_i))^2\end{bmatrix}\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix}$$

$$=\begin{bmatrix}\Delta x & \Delta y\end{bmatrix}C(x,y)\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} \tag{15}$$

where matrix C(x, y) captures the intensity structure of the local ghborhood.

Let λ_1,λ_2 be the eigenvalues of matrix C(x, y). The eigenv s form a rotationally invariant description. There are three cases to be considered

1. If both λ_1, λ_2 are small, so that the local auto-correlation function is flat (i.e., little change in c(x, y) in any direction), the windowed image region is of approximately constant intensity.
2. If one eigenvalue is high and the other is low, so the local auto-correlation function is ridge shaped, then only local shifts in one direction (along the ridge) cause little change in c(x, y) and significant change in the orthogonal direction; this indicates an edge.
3. If both eigenvalues are high, so the local auto-correlation function is sharply peaked, then shifts in any direction will result in a significant increase; this indicates a corner.

3.4.2 View Angle Determination

This section describes view angle determination algorithm. We use Harris corner detector for locating key points of the image. Threshold is selected such as that number of feature points remains from 80 to 100. Key point descriptors are calculated same as in step 3 of stage 1. The descriptors of the key points of the query image are compared with the descriptors of key points of the corresponding database images. One to one comparison is carried out for finding out matching key points. Descriptors of two features are considered as matched when the cosine of angle between their descriptor (equation 12) is above some threshold T_c.

Centre of the image is used as benchmark. Slope of line joining center of the image and key point is used to describe the relative location of the key point. Relative locations of matching key points of query image and database image are compared. The voting strategy is used to select the most likely image for the query image. This image represents the view angle of the query image. The block diagram of this algorithm is given in figure 3.13.

The database for this consists of images of building taken at angles separated by 15 degree. The algorithm assumes the knowledge of the building in the query view from previous stage. The accuracy of this algorithm can be further eased by increasing no of views in the database.

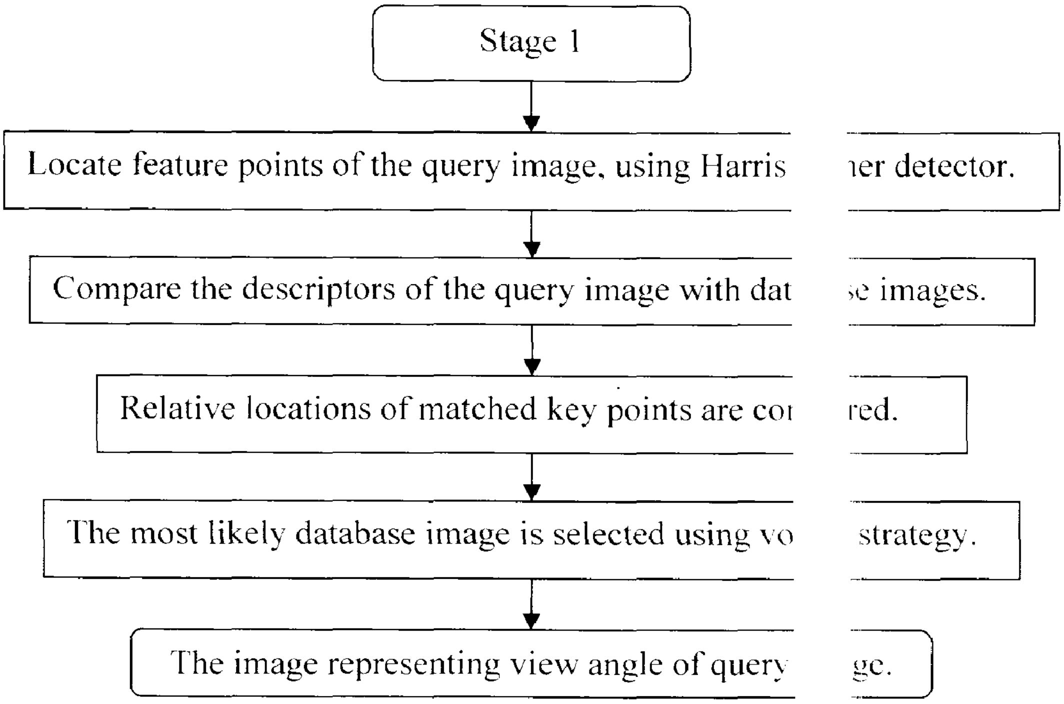


Figure 3.13 Block diagram describing steps in Viewpoint Di ination Algorithm.

4 Implementation Issues

This section discusses about various implementation issues.

4.1 Scalability

Scalability is defined as an ability to handle growing amount of work in graceful manner. While considering scalability of our system, one should think about database size, image size, number of users accessing system.

1. Database size

   We tested out our system with two databases. These databases are described in section 5. Database 1 consists of 210 images and database 2 consists of 1005 images. According to execution time for different steps, noted in table 5.1, we can say that our system is scalable in case of database size. This is achieved because of hierarchical design. First step of building recognition algorithm extracts few possible candidates from entire database by comparing global descriptor of a query image with database images. As global descriptor of an image is 32 dimension vector, time required for this is very less as compared to other processing. And global descriptors of database images are constructed offline and stored in a file. So even if number of images in the database increases, there is no considerable growth in execution time. Second step of building recognition algorithm further narrows the region for search of exact match for query image.

2. Image size

   We tested our system using images of sizes 320 x 240, 640 x 480, 1280 x 1024. Times required with different image sizes are summarized in table 5.2. It shows that time required for each step increases exponentially as image size increases. As our system is an interactive system, response time is very essential. By seeing times required, we can say that image size 640 x 480 is more appropriate for our system.

3. Number of users accessing the system

   Our system is developed in Matlab 7.0. It is a single user system. It can handle query of single user at a time. So as frequency of accessing system increases, we need to upgrade hardware like processor, RAM, to k response time within reasonable limit. As we are not using any special Matl unction, except one or two, our system can be transfer into language like VC where multiple users can be handled in parallel.

4.2 Accuracy

As we mentioned in section 5, we tested our system using databases. Using the database 1 described earlier, we attempted to determine the p of 327 query images. Each query took around 16 s using 3GHz desktop PC. Overall views were registered correctly. Using the database 2 described earlier, we attempte(determine the pose of 115 query images. Each query took around 15 s using 3GHz .ktop PC. Overall 104 views were registered correctly. It highlights the reasonable acc :y of the system.

4.3 Successes

Following are some achievements of our system

1. Hierarchical design of system makes it scalable to data : size. It improves the response time of the system considerably. System can l with large databases easily and with reasonable response time.
2. Algorithm for locating feature points able to deal wi cclusions. It removes clutter and occlusions from image and locates points onl om building part of an image.
3. Clustering method used for matching reduces random m es considerably.
4. Experimental results highlights that view angle de iination algorithm is efficient and reasonably accurate.

4.4 Failures

Global and local features of an image have their pros and cons. ugh system uses both global and local features, clustering method for matching lool iore than local region around feature point; we failed to construct a combine desc r, consisting of both global and local information, of an image. This will further improve accuracy of the system.

4.5 Design Issues

Following are some factors which cause certain design decisions

1. Scalability

As it is an interactive system, it should be able to deal with large database with reasonable response time. For finding out matching image, it was necessary to compare a query image with each database image. But comparison with database image degraded the response time of system. It further degrades as database size increases. So we need a mechanism to reduce the comparison to few probable candidates. We decide to use of global descriptor of an image to locate probable candidates and then use local descriptors to locate exact match. It leads to hierarchical design of system. The hierarchical design also allows us to accommodate both global and local descriptor in recognition process.

2. Efficiency

To improve response time of system we decide to construct global and local descriptors of database images offline and store them in database. Efficiency concern also affects selection of initial value of σ for constructing Gaussian pyramid. We chose $\sigma = 1.2$ by doing trade off between processing time and number of feature points to represent image.

3. Accuracy

Accuracy requirement of system affects selection of descriptor size of local features. It uses GLOH (Gradient Location Orientation Histogram) to describe local features. GLOH generates 272 dimensional vector for each local feature. Finding the nearest neighbour to a query point rapidly becomes inefficient as the dimensionality of the feature space increases. Hash table search is quite efficient in low dimensional spaces, but it is not suitable for high dimensional space. A Best Bin First Search method [18] is also suitable for moderate dimensionality (e.g. 8-15). It forces to use one to one matching feature points of images.

4. Development time

Matlab is suitable for image processing. Development tir :quired is also less for Matlab. It causes the selection of Matlab 7.0 as developme anguage. As Matlab is comfortable with file handling, we chose to use files to stor ta.

4.6 Software Design and Software Development Life Cycl

We used functional programming approach for develop a system. Iterative development prescribes the construction of initially small but :r larger portions of a software project to help to uncover important issues early l re problems or faulty assumptions can lead to disaster. We used Iterative process fo velopment. It helps us to understand strength and drawbacks of previous approaches. illows us to take some design decision during later part of software development cycle

5 Experimental Result and Discussion

The experiments reported in this section were conducted on two different databases. First one (database 1) was constructed by us. It compromises of 42 buildings, with 5 images, acquired from five different view angles, per building. It consists of 210 images of resolution 640 x 480. The database for viewpoint identification algorithm consists of images of resolution 640 x 480 from view angle separated by 15 degree for all 42 buildings. Query images were acquired with large variation of viewpoints, at different times of the day by using the mobile phone camera. Some occlusions by tree, passer-by, and vehicles were purposely included to illustrate the robustness of our approach. Illustrative results are shown in figure 5.1.

The second database (database 2) we used is a ZuBud database [19]. The database comprises of 201 buildings. 5 images per building were acquired with large variation of viewpoints, in different seasons, weather and illumination conditions and by two different cameras. It consists of 1005 images of resolution 640 x 480. Images from different view angle as per requirement for viewpoint identification algorithm are not available. So we test only building recognition algorithm with this database. Query images were acquired with large variation of viewpoints, at different times of the day. Purposely some occlusions by trees and other objects were included in some query images. Illustrative results are shown in figure 5.2.

Using the database 1 described earlier, we attempted to determine the pose of 327 query images. Each query took around 16 s using 3GHz desktop PC. Overall 310 views were registered correctly. Using the database 2 described earlier, we attempted to recognize the location of 115 query images. Each query took around 15 s using 3GHz desktop PC. Overall 104 views were registered correctly.

We measure the execution time for constructing histogram of the image and locating feature points of the image using images with different sizes. Results are summarized in the table 5.1. We measure the execution time for finding o orresponding database image using database 1 and database 2. Results are summariz n the table 5.2. These experiments highlight the accuracy and execution speed of our em. The system works with reasonable speed for low quality images like mobile ca a images. The system fails in case if buildings are too identical in structure and colour

Table 5.1 Execution time for construction colour descriptor of i mage, construction of shape context and locating feature points in the image, and dete ning view angle of the image (out of average 15 views).

	Time Required for (Appı			
Image size	**Colour Histogram Stage 1 step 1**	**Shape Matching Stage 1 Step 2**	**Feature Points Stage 1 Step 3**	**Feature points for Stage 2**
320 x 240	1.15 s	0.95 s	3.05 s	0.60s
640 x 480	3.35 s	2.85 s	6.45 s	1.90s
1280 x 1024	12.75 s	8.75 s	28.15 s	9.40s

Table 5.2 Comparison of execution time using database different sizes

	Time Required for (A ox)	
Stage	**Database 1 with 210 images**	**Datab 2 with 1005 images**
Stage 1 : Step1	3.35 s	3.25 s
Stage 1 : Step2	2.85 s	2.90 s
Stage 1 : Step3	3.95 s	3.45 s
Stage 2	1.90 s	Da se not available

Some of examples highlighting strengths of our system are wn and discussed in figures 5.3-5.7.

Figure 5.1 Illustrative results for database 1.

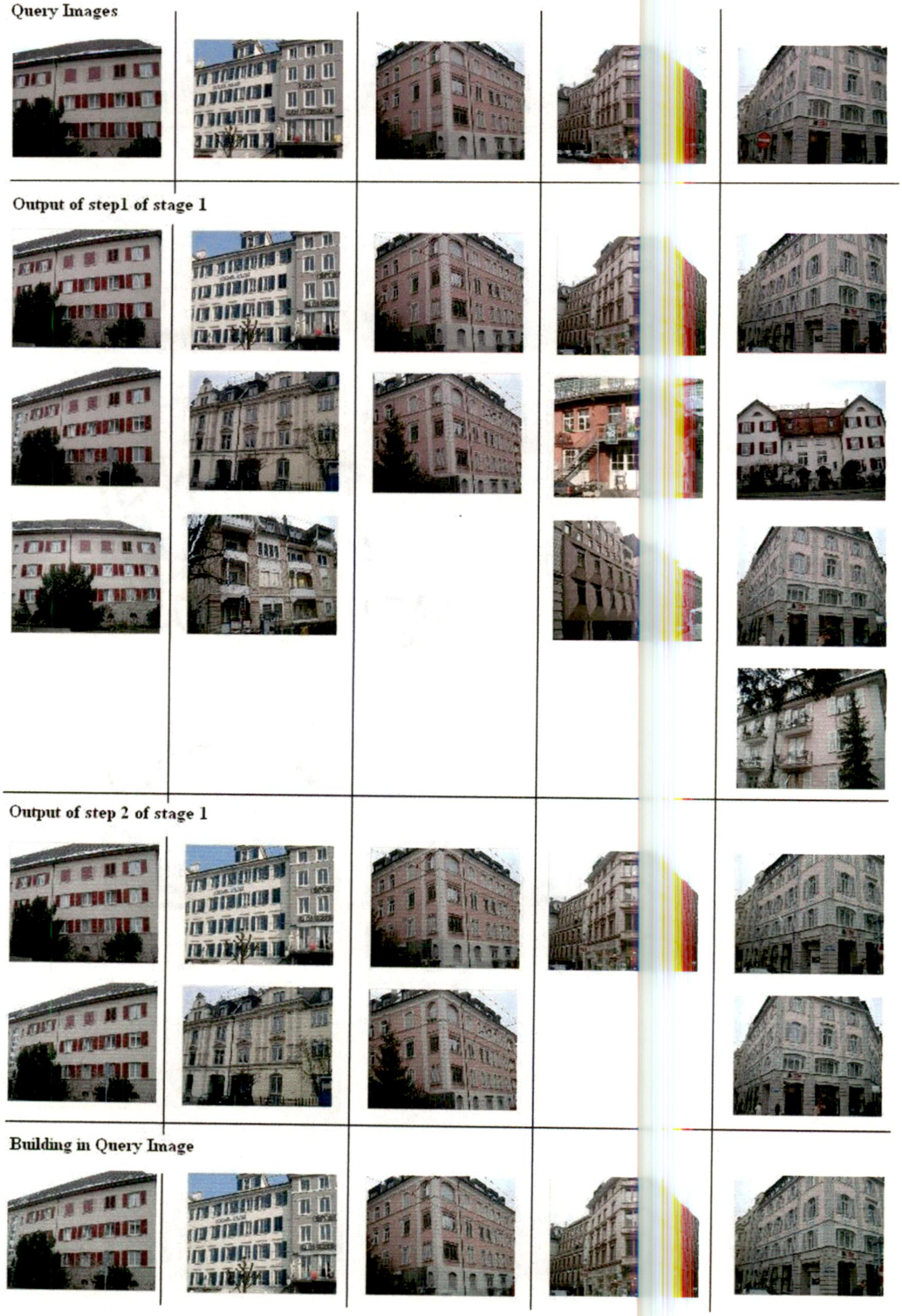


Figure 5.2 Illustrative results for database 2.

Query Image

Output of step 1 of stage 1

Output of step 2 of stage 1

Building in Query Image

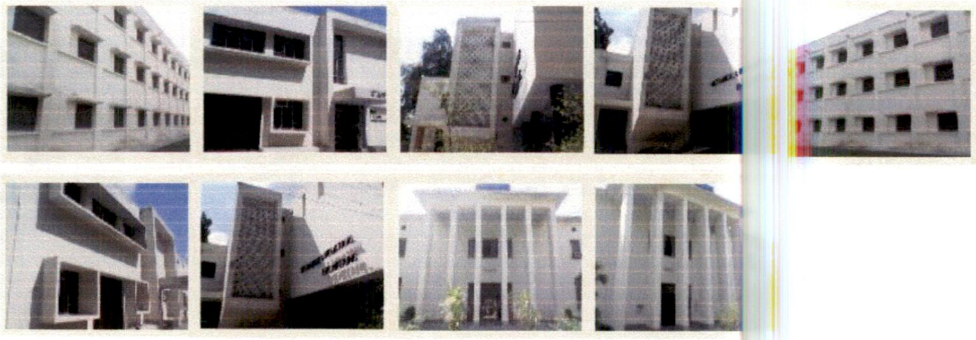
View point of Query Image

Figure 5.3 Query image has considerable occlusion due to tree and passer-by. Figure illustrates accuracy of system in case of occlusion.

Example in figure 5.3 illustrates the ability of system to deal with Occlusions. The query image has considerable occlusions due to tree and passer-by. Database images are snapped with little possible occlusions.

Query Image

Output of step 1 of stage 1



Output of step 2 of stage 1

Building in Query Image

View point of Query Image

Figure 5.4 Query image has snapped at evening time, when light is dull. Figure illustrates accuracy of system in case of intensity variation.

Example in figure 5.4 illustrates the ability of system to deal with intensity variations. The query image has snapped at evening time, when light is dull. Database images are snapped at bright light conditions.

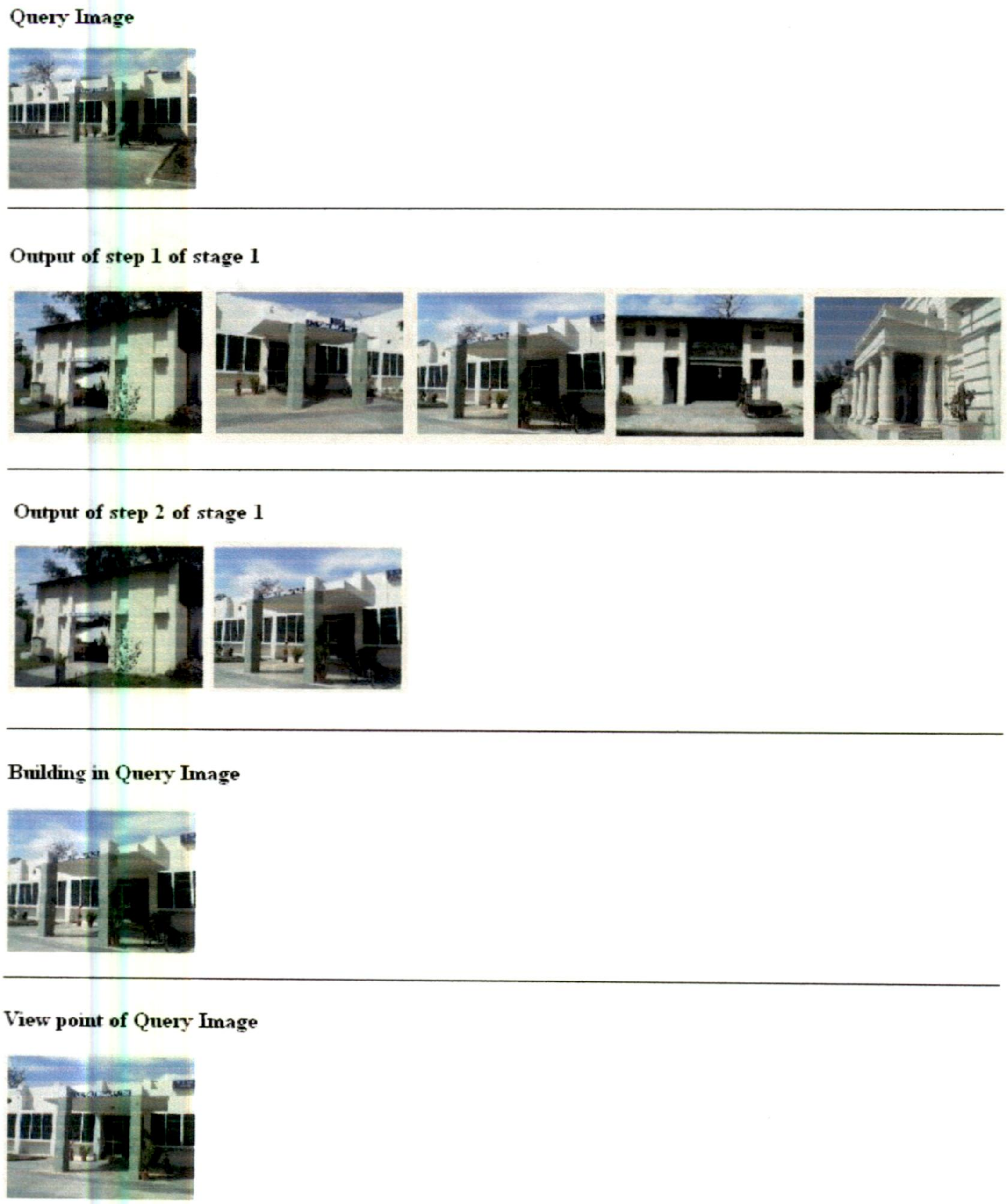


Figure 5.5 Query image has snapped at different scale. Figure illustrates accuracy of system in case of scale variation.

Example in figure 5.5 illustrates the ability of system to deal with scale variation. The query image has snapped from a distance different from distance at which database images have snapped.

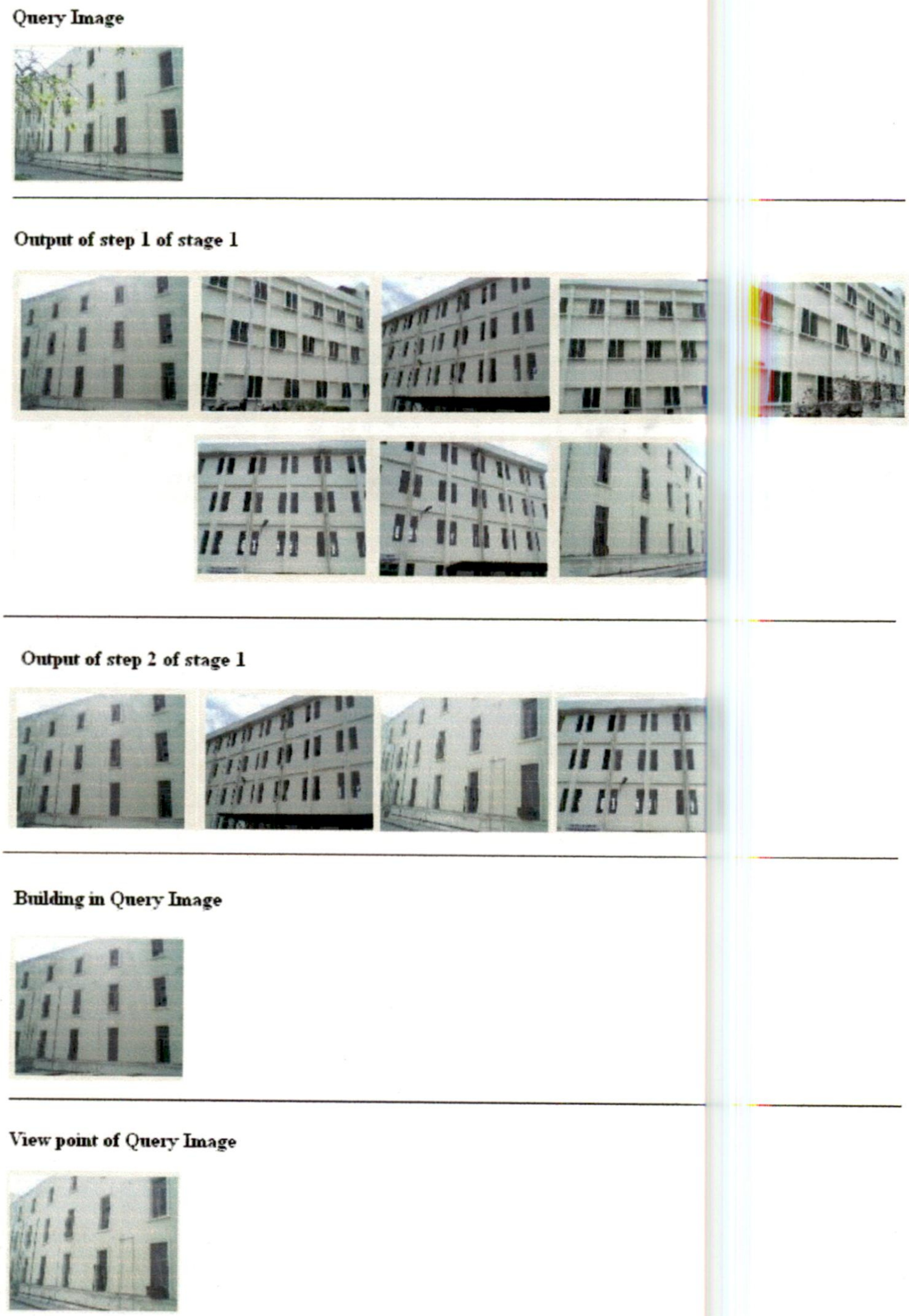


Figure 5.6 Query image has considerable clutter due to tree branches. Figure illustrates accuracy of system in case of clutter.

Example in figure 5.6 illustrates the ability of system to deal with clutter. The query image has considerable clutters due to tree branches. Database images are snapped with little possible clutters.

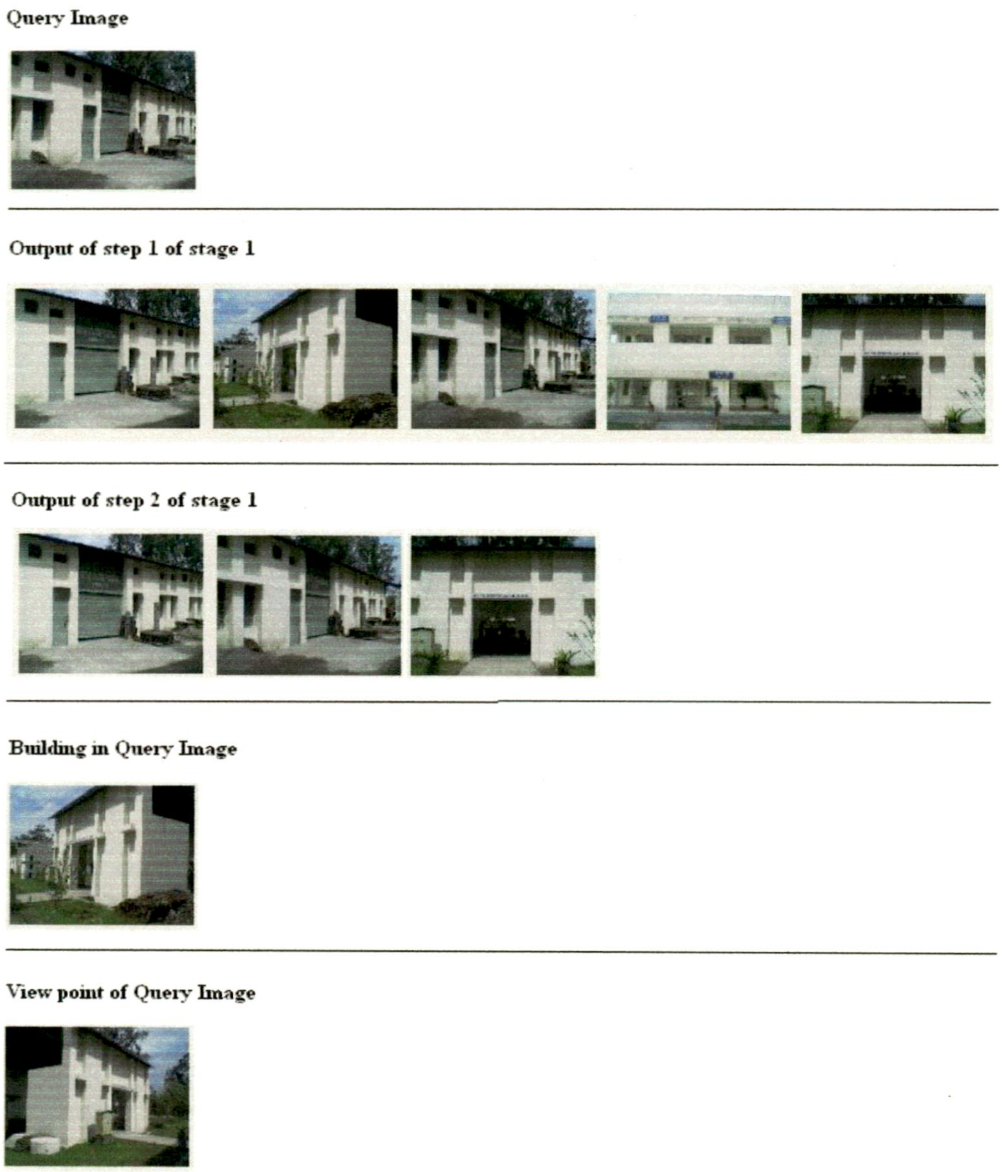


Figure 5.7 Database consists of building with very similar structures. Due to close similar structure, step 3 of stage 1 fails to recognize the building in query image.

Example in figure 5.7 illustrates the possibility of system failure. Because of close similarity between building structures system fails to recognize building in query image.

6 Conclusion and Future Work

In this paper, we have proposed a hierarchical scheme for building recognition. Our experiments illustrate that it has good discrimination capability even for low quality images obtained by cell phone. The colour histogram based first stage reduces the search area. The shape context used in second step of stage one further narrows the search area. The feature points localization in third step of first stage is efficient and able to deal with occlusions and random matching. Descriptors of feature points are more roboust and distinctive. The viewpoint identification algorithm is an efficient technique to identify viewpoint of image. Our experiment shows that overall system works well with low quality query images, acquired by cell phone or PDA. Thus, this method can be used efficiently with large database and for low quality query images, acquired by cell phone or PDA.

Building recognition algorithm fails when buildings are very similar in colour and structure. So the matching algorithm can be further improved by considering additional characteristics of the building such as number of windows etc to recognize them correctly. Algorithm for finding out viewpoint of the query image can be further improved by considering global features of the building image.

References

[1] C. Harris and M.J. Stephens, "A combined corner and edge detector." In Alvey Vision Conference, pp.147–152, 1988.

[2] D. Robertson and R. Cipolla, "An Image-Based System for Urban Navigation", British machine vision conference, pp. 512-518, 2004.

[3] D. Tell and S. Carlsson, "Wide baseline point matching using affine invariants computed from intensity profiles", ECCV, pp. 814-828, 2000.

[4] J. Matas, O. Chum and M. Urban and T. Pajdla, "Robust wide baseline stereo from maximally stable extremal regions", BMVC, pp. 384-396, 2002.

[5] H. Shao, T. Shoboda, T. Tuytelaars and L. Van Gool, "HPAT indexing for fast object/scene recognition based on local appearance", Computer Lecture Notes on Image and Video Retrieval, pp. 71-80, 2003.

[6] S. Coorg and S. Tellar, "Automatic extraction of textured vertical facades from pose imagery", Technical Report TR-759, Massachusetts Institute of Technology, 1998.

[7] T. Goedeme and T. Tuytelaars, "Fast wide baseline matching for visual navigation.", CVPR'04, pages 24-29, 2004.

[8] G. Fritz, C. Seifert, M. Kumar and L. Paletta, "Building Detection from Mobile Phone Imagery Using Informative SIFT Descriptors", Proc. 19th Scandinavian Conference on Image Analysis, SCIA, pp. 629-638, 2005.

[9] P. Pritchett and A. Zisserman, "Wide baseline stereo matching.", ICCV, pp. 863-869, 1998.

[10] W. Zhang and J. Kosecka, "Hierarchical building recognition", Image and Vision Computing, Vol 25, Issue 5, pp: 704-716, May 2007.

[11] D. Lowe, "Distinctive image features from scale-invariant keypoints.", IJCV(60), No. 2, pp. 91-110, 2004.

[12] T. Lindeberg, "Scale-space: A framework for handling image structures at multiple scales", Proc. CERN School of Computing, Egmond aan Zee, The Netherlands, pp 8–21 September, 1996.

[13] J. Kosecka and W. Zhang, "Video compass", Proceedings of European Conference on Computer Vision, pp. 657– 673, 2002.

[14] J. F. Canny, "A computational approach to edge detection", IEEE Trans Pattern Analysis and Machine Intelligence, 8(6), pp. 679-698, Nov 1986.

[15] E. Parzen, "On estimation of a probability density function and mode", Ann. Math. Stat. 33, pp. 1065-1076, 1962.

[16] K. Mikolajczk and C. Schmid, "A performance evaluation of local descriptors", PAMI(27), No. 10, pp.1615-1630, Oct 2005.

[17] M. Trajkovic and M. Hedley, "Fast corner detection", Image and Vision Computing 16, pp 75-87, 1998.

[18] J. Beis and D. Lowe, "Shape indexing using approximate nearest-neighbour search in high-dimensional spaces", Conference on Computer Vision and Pattern Recognition, Puerto Rico, pp 1000-1006, 1997.

[19] T. S. H. Shao and L. V. Gool, "Zubud-zurich buildings database for image based recognition.", Technique report No. 260, Swiss Federal Institute of Technology, 2003.

[20] D. G. Lowe, "Object recognition from local scale-invariant features", International Conference on Computer Vision, Corfu, Greece , pp. 1150-1157, September 1999.

[21] T. Lindeberg and J. O. Eklundh, "On the computation of a scale-space primal sketch", Visual Commun. Image Representation, vol. 2, no. 1, pp. 55-78, 1991.

[22] T. Lindeberg, "Scale-space behavior of local extrema and blobs", Math. Imaging Vision, vol. 1, pp. 65-99, 1992.

[23] C. Xu, and J. L. Prince, "Snakes, Shapes, and Gradient Vector Flow", IEEE Transaction on Image Processing, Vol. 7, No. 3, pp 359-369, March 1998.

[24] L.M.J. Florack, B.M. ter Haar Romeny, J.J. Koenderink, and D M.A. Viergever, "Linear Scale-Space", Journal of Mathematical Imaging and Vision, 4, pp 325-351, 1994.

[25] S. Belongie, J. Malik and J. Puzicha, "Shape Matching and Object Recognition Using Shape Contexts", IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol. 24, No. 24, pp 509-522, April 2002.

[26] S. Belongie, J. Malik and J. Puzicha, "Shape Context: A new descriptor for shape matching and object recognition", NIPS, pp 831-837, 2000.

[27] G. Mori, S. Belongie and J. Malik, "Efficient Shape Matching Using Shape Contexts", PAMI, 27(11), pp 1832-1837, November 2005.

[28] R. Deriche, "Using Canny's criteria to derive an optimal edge detector recursively implemented", International Journal of Computer Vision, Vol. 2, pp 15-20, April 1987.

[29] T. Lindeberg, "Edge detection and ridge detection with automatic scale selection", International Journal of Computer Vision, 30, 2, pp 117-154, 1998.

Appendix A

Source Code

```
%---------------------------------------------------------------------------
%   Locate features of the image
%   Arguments
%        imgpath   :  Image file name
%   Output
%       okflag     :  If 1, this stage is applicable to given image.Otherwise not.
%       hst        :  Hue descriptor of the image
%       shapearray:  Shape context descriptors of the image.
%       fparray    :  Local feature point descriptors.
%       fpimg      :  Image array containing feature points.
%       nlblct     :  No of elements of each component.
%       nlbl       :  No of labels used.
%---------------------------------------------------------------------------
function[okflag,hst,shapearray,fparray,fpimg,nlblct,nlbl] = features(imgpath)
        % return value
        okflag = 1;
        lblct = zeros(1,1);lbl = 1;

        % read image
        colour_img = imread(imgpath);
        %-----------------------------------------------------------------------
        %figure,imshow(climg);title('Original image');
        sz_img = size(colour_img);
        if size(sz_img,2) == 3
            img = double(rgb2gray(colour_img));
        else
            % given image is not color image
            okflag = 2;
            return;
        end
        sz_img = size(img);

        % Detect edges in the image
        fp = edge(img,'canny');
        % Calculate gradient magnitude and direction
```

```
        [grdmag,grdori] = grad_img(img);
        % Find out connected component
        [grp,lblct,lbl] = conn_comp(img,fp,grdmag,grdori);

        % Locate component having length above lenthr
        cmp = zeros( sz_img(1), sz_img(2));
        lenthr = 0.025 * min(sz_img(1),sz_img(2));
        [rl] = find(lblct(1:(lbl-1)) >= lenthr);
        lrl = length(rl);

        % Calculate hue descriptor
        [okflag,hst] = hue_descriptor(colour_img,grp,grdori,rl,lrl);

        % Create shape context descriptors of the image
        [shapearray] = shape_con(grp,grdmag,lrl,rl);

        %  Locate local feature points of the image.
        [fparray,fpimg,nlblct,nlbl] = local_feature_points(img,grdmag,grdori);

%-----------------------------------------------------------------------
%   Calculate gradient magnitude and orientation of the image
%   Arguments
%           img  :  image array
%   Output
%           gradmag  :  Gradient magnitude array.
%           gradori  :  Gradient orientation in degree array.
%-----------------------------------------------------------------------
function[grdmag,grdori] = grad_img(img)

    sz_img = size(img);
    % calculate gradient magnitude and direction
    filter1 = [-1,-2,-1;0,0,0;1,2,1];
    grdx = conv2(img,filter1,'same');
    grdy = conv2(img,filter1','same');
    grdmag = sqrt( grdx .^ 2 + grdy .^ 2 );
    grdori = zeros(sz_img(1),sz_img(2));
    % To avoid divide by zero
    for i = 1 : sz_img(1)
```

```
        for j = 1 : sz_img(2)
            if grdx(i,j) ~= 0
                grdori(i,j) = atand(grdy(i,j)/grdx(i,j));
            else
                if grdy(i,j) < 0
                    grdori(i,j) = -90;
                else
                    grdori(i,j) = 90;
                end
            end
        end
    end

%--------------------------------------------------------------------------
%   Locate connected components of the image.
%   Arguments
%       img      :  Image array.
%       cannyedge:  Image array highlightening edges.
%       grdmag   :  Gradient magnitude array
%       grdori   :  Gradient orinentation in degree array
%   Output
%       grp      :  Image array containing connected components.
%       lblct    :  No of elements of each component.
%       lbl      :  No of labels used.
%--------------------------------------------------------------------------
function[grp,lblct,lbl] = conn_comp(img,cannyedge,grdmag,grdori)

    sz_img = size(img);
    %----------------------------------------------------------------------
    % connected component algorithm
    angthr = 5;                                          %threshold for angle
    % Image mirroring
    grdori = [ grdori(1,1), grdori(1,:), grdori(1,sz_img(2));
               grdori(:,1), grdori,      grdori(:,sz_img(2));
               grdori(sz_img(1),1),
grdori(sz_img(1),:),grdori(sz_img(1),sz_img(2))];
    %Image padding
    cannyedge = [ 0,                    zeros(1,sz_img(2)),0;
```

```
                zeros(sz_img(1),1),cannyedge,        zeros(sz_img(1),1);
                0,                       zeros(1,sz_img(2)),0];
    [r,c] = find(cannyedge);
    lbl = 1;
    noofpts = length(r);
    grp = zeros(sz_img(1) + 2, sz_img(2) + 2);
    lblct = zeros(1,5000);

    for i = 1 : noofpts
        % find lable for each point
        temp = abs(grdori(r(i)-1:r(i)+1,c(i)-1:c(i)+1) - grdori(r(i),c(i)));
        temp = min(180 - temp,temp);

        tempz1 = (temp < angthr) .* cannyedge(r(i)-1:r(i)+1,c(i)-1:c(i)+1);
        % avg orientation
        atemp = grdori(r(i)-1:r(i)+1,c(i)-1:c(i)+1) .* tempz1;
        grdori(r(i),c(i)) = sum(sum(atemp))/9;

        tempz1(2,2) = 0;
        [rl,cl] = find(tempz1);
        rl = rl - 2;cl = cl - 2;
        lrl = length(rl);

        for j = 1 : lrl
            if grp(r(i),c(i)) == 0 && grp(r(i) + rl(j), c(i) + cl(j)) == 0
                %both lable zero
                grp(r(i),c(i)) = lbl;grp(r(i) + rl(j), c(i) + cl(j)) = lbl;
                lblct(lbl) = 2;lbl = lbl + 1;
            elseif grp(r(i),c(i)) ~= 0 && grp(r(i) + rl(j), c(i) + cl(j)) == 0
                % only one of two is zero
                grp(r(i) + rl(j), c(i) + cl(j)) = grp(r(i),c(i));
                lblct(grp(r(i),c(i))) = lblct(grp(r(i),c(i))) + 1;
            elseif grp(r(i),c(i)) == 0 && grp(r(i) + rl(j), c(i) + cl(j)) ~= 0
                % only one of two is zero
                grp(r(i),c(i)) = grp(r(i) + rl(j), c(i) + cl(j));
                lblct(grp(r(i) + rl(j), c(i) + cl(j))) = lblct(grp(r(i) + rl(j),
c(i) + cl(j))) + 1;
            elseif grp(r(i),c(i)) == grp(r(i) + rl(j), c(i) + cl(j))
```

```
                % do nothing
            else
                % both nonzero, then merge group
                if grp(r(i),c(i)) < grp(r(i) + rl(j), c(i) + cl(j))
                    [tr,tc] = find(grp == grp(r(i) + rl(j), c(i) + cl(j)));
                    lblct(grp(r(i),c(i))) = lblct(grp(r(i),c(i))) + lblct(grp(r(i)
+ rl(j), c(i) + cl(j)));
                    lblct(grp(r(i) + rl(j), c(i) + cl(j))) = 0;
                    grp([(tc-1) * (sz_img(1)+2) + tr]) = grp(r(i),c(i));
                elseif grp(r(i),c(i)) > grp(r(i) + rl(j), c(i) + cl(j))
                    [tr,tc] = find(grp == grp(r(i),c(i)));
                    lblct(grp(r(i) + rl(j), c(i) + cl(j))) = lblct(grp(r(i) +
rl(j), c(i) + cl(j))) + lblct(grp(r(i),c(i)));
                    lblct(grp(r(i),c(i))) = 0;
                    grp([(tc-1) * (sz_img(1)+2) + tr]) = grp(r(i) + rl(j), c(i) +
cl(j));
                end
            end
        end
    end
    grp = grp(2:sz_img(1)+1,2:sz_img(2)+1,:);
    lblct = lblct(1:(lbl - 1));

%--------------------------------------------------------------------------
%   Create hue descriptor of the image
%   Arguments
%           color_img:  Color image arrray.
%           grp      :  Image array containing connected components.
%           rl       :  Group numbers having length greater than threshold.
%           lrl      :  No of groups having length greater than threshold.
%   Output
%           okflag   :  If 1, this stage is applicable to given image.Otherwise
not.
%           hst      :  Hue descriptor of the image
%--------------------------------------------------------------------------
function[okflag,hst] = hue_descriptor(colour_img,grp,grdori,rl,lrl)
        % return value
        okflag = 1;
```

```
        hst(32) = 0;
        sz_img = size(grp);

        %-------------------------------------------------------------------
        % Distribute pts into bins according to their orientation. Use 12 bins
        anglebin = [0,0,0,0,0,0,0,0,0,0,0,0];
        for i = 1 : lrl
            [r,c] = find(grp == rl(i));
            pts = sort((c-1) * sz_img(1) + r);
            npts = length(r);
            for j = 1 : npts
                if ( 82.5 < grdori(r(j),c(j)) &&  grdori(r(j),c(j)) <= 90 ) || ( -
90 <= grdori(r(j),c(j)) &&  grdori(r(j),c(j)) <= -82.5 )
                    anglebin(12) = anglebin(12) + 1;
                else
                    angle = -75;k = 0;
                    while ((angle - 7.5) < grdori(r(j),c(j))) && k < 12
                        angle = angle + 15;
                        k = k + 1;
                    end
                    if k ~= 12
                        anglebin(k) = anglebin(k) + 1;
                    end
                end
            end
        end

        %-------------------------------------------------------------------
        % Find out principle direction. Locate peaks in histogram.
        anglebin = [ anglebin(12) anglebin anglebin(1)];
        angle = -75;
        diffp = [180 180];
        pdir = zeros(2,2);
        pdir(2,:) = -1;
        for k = 2 : 13
            if anglebin(k-1) <= anglebin(k) && anglebin(k+1) <= anglebin(k)
                diff = abs(90 - abs(angle));
                pos = (diff < 45) + 1;
```

```
            if diff < diffp(pos)
                pdir(1,pos) = k - 1;
                pdir(2,pos) = angle;
                diffp(pos) = diff;
            end
        end
        angle = angle +  15;
    end

%----------------------------------------------------------------------
%if not a single peak
if pdir(2,1) == -1 && pdir(2,2) == -1
    okflag = 4;
    return;
elseif pdir(2,1) == -1 || pdir(2,2) == -1
    if pdir(2,1) == -1
        k = 1; l = 2;
    elseif pdir(2,2) == -1
        k = 2; l = 1;
    end
    pdir(2,k) = pdir(2,l) + 90;
    if pdir(2,k) > 90
        pdir(2,k) = pdir(2,k) - 180;
    end
    angle = -75;
    while (angle - 7.5) < pdir(2,k) && pdir(1,k) < 13
        angle = angle + 15;
        pdir(1,k) = pdir(1,k) + 1;
    end
end

%----------------------------------------------------------------------
% create hue histogram
temp_imppx = zeros(sz_img(1),sz_img(2));
dir = [-75,-60,-45,-30,-15,0,15,30,45,60,75,90];
cnm = [-0.115 -0.385 0.5; 0.5 -0.454 -0.046];
tempbin = linspace(-1,1,17);
```

```
        for l = 1 : 2
            imppx = zeros(sz_img(1),sz_img(2));
            if pdir(1,l) ~= 12
                imppx = (grdori > (dir(pdir(1,l)) - 7.5)  &  grdori <=
(dir(pdir(1,l))) + 7.5 );
            else
                imppx = (grdori > 82.5  & grdori <= 90) + (grdori >= -90  & grdori
<= -82.5);
            end
            temp_imppx = temp_imppx | imppx;
            % construct histogram of the hue of the pixel along the principle
direction
            [r,c] = find(imppx);
            k = length(r);
            for i = 1:k
                temp = cnm *
double([colour_img(r(i),c(i),1);colour_img(r(i),c(i),2);colour_img(r(i),c(i),3)]);
                huet = (atan2(temp(1,1),temp(2,1)))/pi;
                b = 0;k = -1;
                while (b < 16) && (tempbin(b+1) <= huet)
                    b = b + 1;
                end
                b = (16 * (l-1)) + b;
                hst(b) = hst(b) + 1;
            end
        end
        %normalize histogram
        shst = sum(hst);
        hst = (hst ./ shst) .* 100;
        %-------------------------------------------------------------------
        %figure,bar(1:32,hst);

%-----------------------------------------------------------------------------
%   Create shape context descriptors of the image
%   Arguments
%           ngrp      :  Image array containing connected components.
%         ngrdmag     :  Gradient magnitude array.
```

```
%           rl       :  Group numbers having length greater than threshold.
%           lrl      :  No of groups having length greater than threshold.
%   Output
%          shapearray:  Shape context descriptors of the image
%-----------------------------------------------------------------------
function[shapearray] = shape_con(ngrp,ngrdmag,lrl,rl)
        % return value
        shapearray = zeros(lrl,40);

        for i=1:25  %pad image borders with enough for filter order
            [h,w] = size(ngrdmag);
            ngrdmag = [0,         zeros(1,w),0;
                        zeros(h,1),ngrdmag,  zeros(h,1);
                        0,          zeros(1,w),0];
            ngrp = [0,         zeros(1,w),0;
                    zeros(h,1),ngrp,       zeros(h,1);
                    0,         zeros(1,w),0];
        end

        sz_ngrp = size(ngrp);
        cmp = zeros(sz_ngrp(1),sz_ngrp(2));
        for i = 1 : lrl
            [r,c] = find(ngrp == rl(i));
            % Find center of component
            lenx = max(r) - min(r);
            leny = max(c) - min(c);
            if lenx >= leny
                [sr so] = sort(r);
                mid = floor(length(sr)/2);
                cr = sr(mid);cc = c(so(mid));
                l = floor(lenx/2);
            else
                [sr so] = sort(c);
                mid = floor(length(sr)/2);
                cc = sr(mid);cr = r(so(mid));
                l = floor(leny/2);
            end
           % Upper corner
```

```
            ur = cr - 1;uc = cc - 1;
            if ur < 1
                ur = 1;
            end
            if uc < 1
                uc = 1;
            end
            % Lower corner
            lr = cr + 1 + 1;lc = cc + 1 + 1;
            if lr > sz_ngrp(1)
                lr = sz_ngrp(1);
            end
            if lc > sz_ngrp(2)
                lc = sz_ngrp(2);
            end

             % Construct shape context descriptor
             shapearray(i,1:40) = shape_con_des(ngrp(cr-25:cr+24,cc-
25:cc+24),ngrdmag(cr-25:cr+24,cc-25:cc+24));
             pts = sort((c-1) * sz_ngrp(1) + r);
             cmp(pts) = 255;
        end
%          figure,imagesc(cmp);colormap gray;title('Shapes');

%-----------------------------------------------------------------------------
%   Construct descriptor for shape.
%   Arguments
%            tgrp      :  Shape.
%            tgrdmag   :  Gradient magnitude array for shape.
%   Output
%            shapearray:  Shape context descriptor.
%-----------------------------------------------------------------------------
function[shapearray] = shape_con_des(tgrp,tgrdmag)

    shapearray = zeros(1,40);
    % create mask
    r1 = 25;r2 = 20;r3 = 15;r4 = 10;r5 = 5;
    [x,y]=meshgrid(-r1:(r1-1),-r1:(r1-1));
```

```
    z = zeros(30,30);
    z = ((x.^2+y.^2)<=(r1^2)) + ((x.^2+y.^2)<=(r2^2)) + ((x.^2+y.^2)<=(r3^2)) +
((x.^2+y.^2)<=(r4^2)) + ((x.^2+y.^2)<=(r5^2));

    mk = zeros(50,50);
    ct = 1;
    for i = 5 : -1 : 1
        mk = mk +  ct     * ((z == i) & (x >= 0 & y >= 0) & (     x  <      y ))
+ (ct + 1) * ((z == i) & (x >= 0 & y >= 0) & (     x >=     y ));
        mk = mk + (ct + 2) * ((z == i) & (x >= 0 & y <  0) & (     x  >= abs(y)))
+ (ct + 3) * ((z == i) & (x >= 0 & y <  0) & (     x <  abs(y)));
        mk = mk + (ct + 4) * ((z == i) & (x <  0 & y <  0) & ( abs(x)  <  abs(y)))
+ (ct + 5) * ((z == i) & (x <  0 & y <  0) & ( abs(x) >= abs(y)));
        mk = mk + (ct + 6) * ((z == i) & (x <  0 & y >= 0) & ( abs(x) >=      y ))
+ (ct + 7) * ((z == i) & (x <  0 & y >= 0) & ( abs(x) <      y ));
        ct = ct + 8;
    end

    for i = 1 : 40
        [tr tc] = find(tgrp & (mk == i));
        for j = 1 : length(tr)
            shapearray(1,i) = shapearray(1,i) + tgrdmag(tr(j),tc(j));
        end
    end

%-----------------------------------------------------------------------
%   Locate local feature points of the image.
%   Arguments
%       img      :  Image array.
%       cannyedge:  Image array highlightening edges.
%       grdmag   :  Gradient magnitude array
%       grdori   :  Gradient orinentation in degree array
%   Output
%       fparray  :  Local feature point descriptors.
%       fpimg    :  Image array containing feature points.
%       nlblct   :  No of elements of each component.
%       nlbl     :  No of labels used.
```

```
%------------------------------------------------------------------------------
function[fparray,fpimg,nlblct,nlbl] = local_feature_points(img,grdmag,grdori)
        % return value
        lblct = zeros(1,1);lbl = 1;
        fparray = zeros(1, 275);

        sz_img = size(img);

        % Construct pyramid
        sigma = 1.2;levels = 5;
        pyr = pyramid(img,levels,sigma);

        % Locate edges present at all levels of pyramid.
        fp = ones(sz_img(1),sz_img(2));
        for i = 1 : levels
            fp = fp & edge(pyr{i},'canny');
        end
        fp = (fp > 0);

        % Find out connected component
        [grp,lblct,lbl] = conn_comp(img,fp,grdmag,grdori);

        % Locate component having length above lenthr
        cmp = zeros( sz_img(1), sz_img(2));
        lenthr = 0.025 * min(sz_img(1),sz_img(2));
        [rl] = find( lblct(1:(lbl-1)) >= lenthr);
        lrl = length(rl);

        % Locate feature points
        fpimg = img;
        nlblct = zeros(1,lrl);
        nlbl = lrl;
        for i = 1 : lrl
            [r,c] = find(grp == rl(i));
            pts = sort((c-1) * sz_img(1) + r);
            cmp(pts) = 255;
            fpimg(pts(1:5:length(pts))) = 255;
```

```
        nlblct(i) = length(1:5:length(r));
    end
    [r,c] = find(cmp);

    % Parzen Window Approach
    if length(r) > 500
        tcmp = cmp;
        p = 0.2;
        e = 9;sg = 2 * (e/3)^2;
        [x,y]=meshgrid(-e:(e-1),-e:(e-1));
        z = ((x.^2+y.^2)<=(e^2));z(e+1,e+1) = 0;
        % image padding
        for j = 1 : e
            tsz = size(tcmp);
            tcmp = [ 0,                 zeros(1,tsz(2)), 0;
                     zeros(tsz(1),1),   tcmp,            zeros(tsz(1),1);
                     0,                 zeros(1,tsz(2)), 0 ];
        end
        [r,c] = find(tcmp);
        for i = 1 : length(r)
            tmp = tcmp(r(i)-9:r(i)+8,c(i)-9:c(i)+8);
            tmp = tmp & z;
            [rz,cz] = find(tmp);
            w = 0;
            for j = 1 : length(rz)
                w = w + exp(-1 *((x(rz(j),cz(j))^2 + y(rz(j),cz(j))^2)/(sg)));
            end
            tcmp(r(i),c(i)) = w;
        end
        tsz = size(tcmp);
        tcmp1 = (tcmp(10:tsz(1)-9,10:tsz(2)-9)) > p;
        cmp = cmp .* tcmp1;
    end
%       figure,imagesc(cmp);colormap gray;title('Local features points');

    % Construct descriptors for feature points.
    [fparray] = local_feature_points_des(sz_img,cmp,grdmag,grdori,1);
```

```
%------------------------------------------------------------------------
%   Construct descriptors for feature points.
%   Arguments
%            sz_img   :  Input image size.
%            cmp      :  Feature points of image.
%            grdmag   :  Gradient magnitude array.
%            grdori   :  Gradient orientation array.
%            v        :  If v = 1, construct descriptor for step 3 of stage1.
%                        If v = 2, construct descriptor for stage2.
%   Output
%            fparray  :  Feature point descriptor array.
%------------------------------------------------------------------------
function[fparray] = local_feature_points_des(sz_img,cmp,grdmag,grdori,v)
    fparray = zeros(1, 275);

    %-------------------------------------------------------------------
    % construct descriptors
    % construct mask
    r1 = 15;r2 = 11;r3 = 6;
    [x,y]=meshgrid(-r1:(r1-1),-r1:(r1-1));
    z = zeros(30,30);
    z = ((x.^2+y.^2)<=(r1^2)) + ((x.^2+y.^2)<=(r2^2)) + ((x.^2+y.^2)<=(r3^2));

    mk = (z == 3);
    ct = 2;
    for i = 2 : -1 : 1
        mk = mk +  ct      * ((z == i) & (x >= 0 & y >= 0) & (      x  <       y ))
+ (ct + 1) * ((z == i) & (x >= 0 & y >= 0) & (      x >=      y ));
        mk = mk + (ct + 2) * ((z == i) & (x >= 0 & y <  0) & (      x  >= abs(y)))
+ (ct + 3) * ((z == i) & (x >= 0 & y <  0) & (      x <  abs(y)));
        mk = mk + (ct + 4) * ((z == i) & (x <  0 & y <  0) & ( abs(x) <  abs(y)))
+ (ct + 5) * ((z == i) & (x <  0 & y <  0) & ( abs(x) >= abs(y)));
        mk = mk + (ct + 6) * ((z == i) & (x <  0 & y >= 0) & ( abs(x) >=      y ))
+ (ct + 7) * ((z == i) & (x <  0 & y >= 0) & ( abs(x) <       y ));
        ct = ct + 8;
```

```
    end

    [r,c] = find(cmp);
    noofpts = length(r);
    if v == 1
        % three extra to store lable and location of feature point
        fparray = zeros(noofpts, 275);
    else
        % two extra to store location of feature point
        fparray = zeros(noofpts, 274);
    end

    fpct = 1;
    for i = 1 : noofpts
        if r(i) > r1 && c(i) > r1 && r(i) < sz_img(1) - (r1-1) && c(i) < sz_img(2)
- (r1-1)
            tgrdmag = grdmag(r(i) - r1 : r(i) + (r1-1),c(i) - r1 : c(i) + (r1-1));
            tgrdori = grdori(r(i) - r1 : r(i) + (r1-1),c(i) - r1 : c(i) + (r1-1));

            ct = 1;
            for id = 1 : 17
                tmk = (mk == id);
                fparray(fpct,ct : ct + 15) = des((tgrdmag .* tmk),(tgrdori .*
tmk));
                ct = ct + 16;
            end

            % normalize vector
            temp = fparray(fpct,1:272);
            temp = temp / sqrt(sum(temp.^2));
            fparray(fpct,1:272) = temp(1,1:272);

            if v == 1
                % store lable of feature point
                fparray(fpct,273) = cmp(r(i),c(i));
                fparray(fpct,274) = r(i);fparray(fpct,275) = c(i);
            else
```

```
                % store location of feature point
                fparray(fpct,273) = r(i);fparray(fpct,274) = c(i);
            end
            fpct = fpct + 1;
        end
    end

    %------------------------------------------------------------------------
    %   Construct descriptors for given feature point.
    %   Arguments
    %           grdmagtemp :  Gradient magnitude array in neighbourhood of feature
point.
    %           grdmagtemp :  Gradient orientation array in neighbourhood of
feature point.
    %   Output
    %           fparray    :  Feature point descriptor array.
    %------------------------------------------------------------------------
    function[fparray] =  des(grdmagtemp, grdoritemp)
        fparray = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
        [k,l] = find(grdmagtemp);
        npts = length(k);
        for m = 1 : npts
            ang = -90;b = 0;
            while b < 16 && grdoritemp(k(m),l(m)) >= ang
                b = b + 1;
                ang = ang + 11.25;
            end
            fparray(b) = fparray(b) + grdmagtemp(k(m),l(m));
        end

%----------------------------------------------------------------------------
%   Store image and its features in the database.
%   Arguments
%        databasePath:  Database path.
%           filename :  Image file name.
%           bldno    :  Number assigned to building in image.
%           hst      :  Colour histogram.
%         shapearray:  Shape context descriptors of the image.
```

```
%             fparray  :  Local feature point descriptors.
%             lbl      :  No of labels used.
%             lblct    :  No of elements of each component.
%   Output
%----------------------------------------------------------------------------
function
store_features(databasePath,filename,bldno,hst,shapearray,fparray,lbl,lblct)
    % Read no of files
    no_of_images = 0;
    fid = fopen([databasePath '\' 'db_image_names.dat'],'r');
    [nimages count] = fscanf(fid,'%s',[1 1]);
    no_of_images = str2num(nimages);
    fclose(fid);

    fid = fopen([databasePath '\' 'db_image_names.dat'],'r+');
    no_of_images = no_of_images + 1;
    nimages = ['0000' num2str(no_of_images)];
    nimages = nimages(length(nimages) - 4:length(nimages));
    fprintf(fid,'%s ',nimages);
    fclose(fid);

    fid = fopen([databasePath '\' 'db_image_names.dat'],'a');
    fprintf(fid,' %d',[bldno]);
    fprintf(fid,' %s',['image' nimages '.mat']);
    fclose(fid);

    % Store colour descriptor
    fid = fopen([databasePath '\' 'db_image_des'],'a');
    fwrite(fid,hst,'double');
    fclose(fid);

    % Store file
    img = imread(filename);
    imwrite(imread(filename),[databasePath '\' 'image' nimages '.jpg'],'jpg');

    % Store local feature points
    fid = fopen([databasePath '\' 'image' nimages '.mat'],'a');
```

```
    % Write building number
    fwrite(fid,bldno,'int32');
    % Write size of shapearray i.e. no of shapes
    fwrite(fid,size(shapearray,1),'double');
    % Store shape array
    fwrite(fid,shapearray,'double');
    % Write size of fparray i.e. no of feaure points
    fwrite(fid,size(fparray,1),'double');
    % Store feature point array
    fwrite(fid,fparray,'double');
    % Store lbl and lblct
    fwrite(fid,lbl,'double');
    fwrite(fid,lblct,'double');
    fclose(fid);

%---------------------------------------------------------------------------
%   Harris corner detector.
%   Arguments
%           imgpath  :  Image file name.
%   Output
%           I        : Image with highlighted feature points.
%           fparray  :  Feature point descriptor array.
%---------------------------------------------------------------------------
function[I,fparray] =  Harris_detector(imgpath)

    img = rgb2gray(imread(imgpath));
    I =double(img);

    cmin= 1;cmax = size(I,1); rmin = 1; rmax = size(I,2);
    % Number of points are limited between min_n and max_N
    min_N=80;max_N=100;

    sigma=2; Thrshold=20; r=6; disp=1;
    % Calculate gradient
    dx = [-1 0 1; -1 0 1; -1 0 1]; % The Mask
    dy = dx';
```

```
sz_img = size(I);
fp = ones(sz_img(1),sz_img(2));

k = 0.04;sze = 2*r+1;
Ix = conv2(I, dx, 'same');
Iy = conv2(I, dy, 'same');
% Gaussien Filter
g = fspecial('gaussian',max(1,fix(6*sigma)), sigma);
Ix2 = conv2(Ix.^2, g, 'same');
Iy2 = conv2(Iy.^2, g, 'same');
Ixy = conv2(Ix.*Iy, g,'same');

R11 = (Ix2.*Iy2 - Ixy.^2) - k*(Ix2 + Iy2).^2;
R11=(1000/max(max(R11)))*R11;
R=R11;
ma=max(max(R));

MX = ordfilt2(R,sze^2,ones(sze));
R11 = (R==MX)&(R>Thrshold);
count=sum(sum(R11(5:size(R11,1)-5,5:size(R11,2)-5)));

loop=0;
while (((count<min_N)|(count>max_N))&(loop<20))
    if count>max_N
        Thrshold=Thrshold*1.5;
    elseif count < min_N
        Thrshold=Thrshold*0.5;
    end
    R11 = (R==MX)&(R>Thrshold);
    count=sum(sum(R11(5:size(R11,1)-5,5:size(R11,2)-5)));
    loop=loop+1;
end

R=R*0;
R(5:size(R11,1)-5,5:size(R11,2)-5)=R11(5:size(R11,1)-5,5:size(R11,2)-5);
fp = fp & R;
```

```
    fp = [0                    zeros(1,sz_img(2));
          zeros(sz_img(1),1)  fp];
    fp = fp(1:size(fp,1)-1,1:size(fp,2)-1);

    % For displaing feature points
    [r1,c1] = find(fp);
    pts = c1 * sz_img(1) + r1;
    I = img;
    I(pts) = 255;
%     figure,imshow(uint8(I));

    % Calculate feature point descriptor
    [grdmag,grdori] = grad_img(double(img));
    [fparray] = local_feature_points_des(sz_img,fp,grdmag,grdori,2);

%------------------------------------------------------------------------
%   Store image and its features ( for angle determination) in the database.
%   Arguments
%        databasePath:  Database path.
%           filename :  Image file name.
%           bldno    :  Number assigned to building in image.
%           fparray  :  Local feature point descriptors.
%   Output
%------------------------------------------------------------------------
function store_features_angle(databasePath,filename,bldno,fparray)

    % Read no of files
    no_of_images = 0;
    fid = fopen([databasePath '\' 'db_image_names_ac.dat'],'r');
    [nimages count] = fscanf(fid,'%s',[1 1]);
    no_of_images = str2num(nimages);
    fclose(fid);

    fid = fopen([databasePath '\' 'db_image_names_ac.dat'],'r+');
    no_of_images = no_of_images + 1;
    nimages = ['0000' num2str(no_of_images)];
    nimages = nimages(length(nimages) - 4:length(nimages));
```

```
    fprintf(fid,'%s ',nimages);
    fclose(fid);

    fid = fopen([databasePath '\' 'db_image_names_ac.dat'],'a');
    fprintf(fid,' %d %s',bldno,['image_ac' nimages '.mat']);
    fclose(fid);

    % Store file
    img = imread(filename);
    imwrite(imread(filename),[databasePath '\' 'image_ac' nimages '.jpg'],'jpg');

    % Store feature points
    fid = fopen([databasePath '\' 'image_ac' nimages '.mat'],'a');
    % Write size of fparray i.e. no of feaure points
    fwrite(fid,size(fparray,1),'double');
    % Store feature point array
    fwrite(fid,fparray,'double');
    fclose(fid);

%-----------------------------------------------------------------------------
%   Stage1 step1 : Compare hue descriptor of the query image with hue descriptor
of database images
%   Arguments
%           databasePath    :  Path to databse.
%           query_image_des :  Hue descriptor of query image.
%   Output
%           stage1img       :  Set of databse images close to query image.
%           stage1bno       :  Set of databse images building number.
%-----------------------------------------------------------------------------
function[stage1img,stage1bno] = match_huedes(databasePath,query_image_des)

    % ---------------------------------------------------------------------------
    % Read database image file names
    no_of_images = 0;
    fid = fopen([databasePath '\' 'db_image_names.dat'],'r');
    [nimages count] = fscanf(fid,'%s',[1 1]);
    no_of_images = str2num(nimages);
```

```
    for i = 1 : no_of_images
        [db_image_names_no(i) count] = fscanf(fid,'%d',[1 1]);
        [db_image_names{i} count] = fscanf(fid,'%s',[1 1]);
    end
    fclose(fid);

    % ---------------------------------------------------------------------
    % Read hue descriptors of database images from file and Compare hue
descriptors.
    comparison = zeros(1,no_of_images);
    fid = fopen([databasePath '\' 'db_image_des'],'r');
    for i = 1 : no_of_images
        [db_image_des(1:32) count] = fread(fid,[1 32],'double');
        tsum = db_image_des(1,:) + query_image_des(:)';
        tmul = ((db_image_des(1,:) - query_image_des(:)').^2);
        for j = 1: 32
            if tsum(j) ~= 0
                comparison(i) = comparison(i) + (tmul(j)/ tsum(j));
            end
        end
    end
    fclose(fid);

    % ---------------------------------------------------------------------
    % Comparison
    cmin = min(comparison);
    nm = 30;
    if length(comparison) < 30
        nm = length(comparison);
    end
    csum = sort(comparison);
    csum = sum(csum(2:nm));

    nr = nm * (((cmin)/( (1/(nm - 1)) * (csum) ))^2);
    nr = ceil(nr);
    if nr > nm
        nr = nm
```

```
    end
    if nr == 0
        nr = 1;
    end

    mx = max(comparison);
    for i = 1 : nr
        [mn mnind] = min(comparison);
        comparison(mnind) = mx;
        stage1img{i} = db_image_names{mnind};
        stage1bno(i) = db_image_names_no(mnind);
    end

%---------------------------------------------------------------------------
%   Stage1 step2 : Compare shapes of the query image with shapes of database
images
%   Arguments
%           databasePath     :  Path to databse.
%           stage1img        :  Output of stage1 step1.
%           stage1bno        :  Output of stage1 step1.
%           qryshapearray    :  Shape descriptor of query image.
%   Output
%           stage2img        :  Set of databse images close to query image.
%---------------------------------------------------------------------------
function[stage2img] = match_shapes(databasePath,stage1img,stage1bno,qryshapearray)

    no_of_images = 0;
    fid = fopen([databasePath '\' 'db_image_names.dat'],'r');
    [nimages count] = fscanf(fid,'%s',[1 1]);
    no_of_images = str2num(nimages);
    for i = 1 : no_of_images
        [db_image_names_no(i) count] = fscanf(fid,'%d',[1 1]);
        [db_image_names{i} count] = fscanf(fid,'%s',[1 1]);
    end
    fclose(fid);

    nr = length(stage1bno);
```

```
stage1bno = sort(stage1bno);
no_img = 0;tbno = 0;
for i = 1 : nr
    if tbno ~= stage1bno(i)
        tbno = stage1bno(i);
        temp = find(db_image_names_no == stage1bno(i));
        for j = 1 : length(temp);
            no_img = no_img + 1;imgs{no_img} = db_image_names{temp(j)};
        end
    end
end

sc = zeros(no_img,1);
qrynoofpts = size(qryshapearray,1);
for i = 1 : no_img
    % -----------------------------------------------------------------
    filename = imgs{i};
    filename = [databasePath '\' filename(1:size(filename,2)-3) 'mat'];
    fid = fopen(filename,'r');

    % Read building number
    [bno count] = fread(fid,[1 1],'int32');
    % Read shape context
    [no_of_fp count] = fread(fid,[1 1],'double');
    [dbshapearray count] = fread(fid,[no_of_fp 40],'double');
    fclose(fid);

    % -----------------------------------------------------------------
    % Compare images with query
    mat = 0;
    dbnoofpts = size(dbshapearray,1);
    for q = 1 : qrynoofpts
        mxang = 0;
        temp_mxang = zeros(1,dbnoofpts);
        for d = 1 : dbnoofpts
            tsum = dbshapearray(d,:) + qryshapearray(q,:);
            tmul = ((dbshapearray(d,:) - qryshapearray(q,:)).^2);
```

```
            for j = 1: 40
                if tsum(j) ~= 0
                    temp_mxang(d) = temp_mxang(d) + (tmul(j)/ tsum(j));
                end
            end
        end
        [mxang mxpos] = max(temp_mxang);
        if mxang >= 0.95
            mat = mat + 1;
        end
    end
    sc(i) = mat/qrynoofpts;
end

% ----------------------------------------------------------------------
% Comparison
sc = -sc;
cmin = min(sc);
nm = no_img;
csum = sort(sc);
csum = sum(csum(2:nm));

nr = nm * (((cmin)/( (1/(nm - 1)) * (csum) ))^2);
nr = ceil(nr);
if nr > nm
    nr = nm
end
if nr == 0
    nr = 1;
end

mx = max(sc);
for i = 1 : nr
    [mn mnind] = min(sc);
    sc(mnind) = mx;
    stage2img{i} = db_image_names{mnind};
end
```

```
%--------------------------------------------------------------------------
%   Stage1 step 3 : Compare the feature points of the query image with database
images.
%   Arguments
%           databasePath    :  Path to databse.
%           stage2img       :  Output of stage1 step2.
%           qryfparray      :  Feature points of the query image.
%           qrylblct        :  No of elements of each component.
%           qrylbl          :  No of labels used.
%   Output
%           bestmatch       :  Best matching image for query image.
%           bestmatch_bno   :  Best matching building number.
%--------------------------------------------------------------------------
function[bestmatch, bestmatch_bno] =
match_local_features(databasePath,stage2img,qryfparray,qrylblct,qrylbl)

    qrynoofpts = size(qryfparray,1);
    nr = size(stage2img,2);
    perc = zeros(nr,2);
    for i = 1 : nr
        % ---------------------------------------------------------------
        filename = stage2img{i};
        filename = [databasePath '\' filename(1:size(filename,2)-3) 'mat'];
        fid = fopen(filename,'r');

        % Read building number
        [bno count] = fread(fid,[1 1],'int32');

        % Read shapearray
        [temp1 count] = fread(fid,[1 1],'double');
        [temp2 count] = fread(fid,[temp1 40],'double');

        % Read fparray
        [no_of_fp count] = fread(fid,[1 1],'double');
        [dbfparray count] = fread(fid,[no_of_fp 275],'double');
```

```
        % Read lbl
        [dblbl count] = fread(fid,[1 1],'double');
        % Read lblct
        [dblblct count] = fread(fid,[1 dblbl],'double');

        fclose(fid);

        % ----------------------------------------------------------------
        % Compare images
        [totalmatch] =
match_local_features_cl(qryfparray,qrylblct,qrylbl,dbfparray,dblblct,dblbl)

        if qrynoofpts ~= 0
             perc(i,1) = totalmatch;
        end
        perc(i,2) = bno;
        % ----------------------------------------------------------------
        %display([db_image_names(stage2img(i),:)  ' : '  num2str(perc(i))]);
    end
    % Best match for query image.
    [mx mxind] = max(perc(:,1));
    bestmatch = stage2img{mxind};
    bestmatch_bno = perc(mxind,2);

%--------------------------------------------------------------------
%   Compare the feature points of the query image with database image.
%   Arguments
%       qryfparray:  Feature points of query image.
%       qrylblct   :  No of elements of each component.
%       qrylbl     :  No of labels used.
%       dbfparray :  Feature points of database image.
%       dblblct    :  No of elements of each component.
%       dblbl      :  No of labels used.
%   Output
%       totalmatch:  No of points matched.
%--------------------------------------------------------------------
```

```
function[totalmatch] =
match_local_features_cl(qryfparray,qrylblct,qrylbl,dbfparray,dblblct,dblbl)

    qrynoofpts = size(qryfparray,1);
    prc = zeros(qrynoofpts,7);
    mat = 0;
    dbnoofpts = size(dbfparray,1);
    dbfparrayt = dbfparray(:,1:272)';
    for q = 1 : qrynoofpts
        temp_mxang = qryfparray(q,1:272) * dbfparrayt;
        [vals,indx] = sort(temp_mxang);  % Take inverse cosine and sort results
        if vals(1) >= 0.9
            mat = mat + 1;

            prc(mat,1) = qryfparray(q,273);
            prc(mat,2) = dbfparray(indx(1),273);
            prc(mat,3) = mxang;

            prc(mat,4) = qryfparray(q,274);
            prc(mat,5) = qryfparray(q,275);
            prc(mat,6) = dbfparray(indx(1),274);
            prc(mat,7) = dbfparray(indx(1),275);
        end
    end

    totalmatch = 0;
    qryclst = prc(:,1);
    dbclst = prc(:,2);
    matwt = prc(:,3);

    prg = zeros(qrylbl,4);
    for cl = 1 : qrylbl
        clloc = find(qryclst == cl);
        matchct = zeros(1,dblbl);
        for i = 1 : length(clloc)
            matchct(dbclst(clloc(i))) = matchct(dbclst(clloc(i))) + 1;
        end
```

```
        [dbclct,dbcl] = max(matchct);

        tdbclloc = find(dbclst == dbcl);
        if dbclct > 0
             if dbclct > (0.5 * qrylblct(cl))
                totalmatch = totalmatch + qrylblct(cl);
%                 totalmatch = totalmatch + dbclct;
                prg(cl,1) = prc(clloc(1),4);prg(cl,2) = prc(clloc(1),5);
                prg(cl,3) = prc(tdbclloc(1),6);prg(cl,4) = prc(tdbclloc(1),7);
             end
        end
    end

%---------------------------------------------------------------------------
% Stage2 : Compare the feature points of the query image with database images.
% Arguments
%     databasePath    :  Path to databse.
%     qryfparray      :  Feature points of the query image.
%     qimg            :  Query Image.
%     bestmatch_bno   :  Building number of query image.
% Output
%     viewangle       :  Best matching image for query image.
%---------------------------------------------------------------------------
function[viewangle] = match_hcfp(databasePath,qryfparray,qimg,bestmatch_bno)

    qrynoofpts = size(qryfparray,1);
    % ----------------------------------------------------------------------
    % read database image file names
    no_of_images = 0;
    fid = fopen([databasePath '\' 'db_image_names_ac.dat'],'r');
    [nimages count] = fscanf(fid,'%s',[1 1]);
    no_of_images = str2num(nimages);
    for i = 1 : no_of_images
        [bno(i) count] = fscanf(fid,'%d',[1 1]);
        [db_image_names{i} count] = fscanf(fid,'%s',[1 1]);
    end
    fclose(fid);
```

```
tbno = find(bno == bestmatch_bno);
ltbno = length(tbno);

bim = 1;
for j = 1 : ltbno
    i = tbno(j);
    %------------------------------------------------------------------
    filename = [databasePath '\' db_image_names{i}];
    fid = fopen(filename,'r');
    % Read size of fparray
    [dbnoofpts count] = fread(fid,[1 1],'double');
    % Read fparray
    [dbfparray count] = fread(fid,[dbnoofpts 274],'double');
    fclose(fid);

    % ----------------------------------------------------------------
    % Compare images
    [matches,prc] = match_hc_features(qryfparray,dbfparray)

    % Wrt center of image
    cx = size(qimg,1)/2;cy = size(qimg,2)/2;
    slope1 = zeros(matches,2);slope2 = zeros(matches,2);
    for j = 1 : matches
        temp = prc(j,1) - cx;
        if temp ~= 0
           slope1(j,1) = atan2((prc(j,2) - cy),temp);
        end
        temp = prc(j,3) - cx;
        if temp ~= 0
           slope2(j,1) = atan2((prc(j,4) - cy),temp);
        end
    end
    % Compare slopes
    mat = 0;
    for k = 1 : matches
```

```
            if slope1(k,1) > slope2(k,1) - 0.15 && slope1(k,1) < slope2(k,1) +
0.15
                mat = mat + 1;
            end
        end
        if matches == 0
            perc(bim,1) = 0;
        else
            perc(bim,1) = (mat/matches) * 50;
        end
        perc(bim,1) = perc(bim,1) + (matches/qrynoofpts) * 50;
        perc(bim,2) = i;
        bim = bim + 1;
    end
    % View angle for query image.
    [mx mxind] = max(perc(:,1));
    viewangle = db_image_names{perc(mxind,2)};

%-------------------------------------------------------------------------------
%   Compare the feature points of the query image with database.
%   Arguments
%           qryfparray  :  Feature points of query image.
%           dbfparray   :  Feature points of database image.
%   Output
%           mat         :  No of points matched.
%           prc         :  Pair of matching points.
%-------------------------------------------------------------------------------
function[mat,prc] = match_hc_features(qryfparray,dbfparray)
    qrynoofpts = size(qryfparray,1);
    mat = 0;
    prc = zeros(noofpts,4);
    dbnoofpts = size(dbfparray,1);

    dbfparrayt = dbfparray(:,1:272)';
    for q = 1 : qrynoofpts
        temp_mxang = qryfparray(q,:) * dbfparrayt;
        [vals,indx] = sort(temp_mxang);  % Take inverse cosine and sort results
        if vals(1) >= 0.9
```

```
            mat = mat + 1;
            prc(mat,1) = qryfparray(q,273);
            prc(mat,2) = qryfparray(q,274);
            prc(mat,6) = dbfparray(indx(1),273);
            prc(mat,7) = dbfparray(indx(1),274);
        end
    end

%---------------------------------------------------------------------------
%   Create laplacian pyramid of the image.
%   Arguments
%           img      :  Image intensity array.
%           levels   :  No of levels required in pyramid.
%           scl      :  scale factor used at level 0.
%   Output
%           pyr      : Cell containing the laplacian pyramid of the image.
%---------------------------------------------------------------------------
function [pyr] = pyramid(img,levels,scl)
    %sigma  = scl;                              % variance for laplacian filter
    sigma  = 1/(scl);
    for i=1:levels
         A = f_gs(img,floor(sigma * 6),sigma);% calculate difference of gaussians
         B = f_gs(A,floor(sigma * 6),sigma);
         pyr{i} = A-B;                          % store result in cell array
         %sigma = scl ^ 2;
         sigma = sigma * scl;
    end
    %-----------------------------------------------------------------------
%     show_pyramid(pyr,levels)  %show pyramid if desired


    %-----------------------------------------------------------------------
    % To display laplacian pyramid of the image.
    function show_pyramid(pyr,levels)
        close all
        [h,w] = size(pyr);
        figure;
        j = 1;l = 0;c = ceil(levels/2);
        for i=1:w
```

```
        subplot(2,c,l * c + j);imagesc(pyr{i});colormap gray;
        j = j + 1;
        if j > c
            j = 1;
            l = l + 1;
        end
    end

%---------------------------------------------------------------------------
%   Apply gaussian filter to given image.
%   Arguments
%           img         :  Image intensity array.
%           order       :  size of the mask.
%           sig         :  scale factor.
%   Output
%           img_out        : Cell containing the laplacian pyramid of the image.
%---------------------------------------------------------------------------
function [img_out] = f_gs(img,order,sig)
    for i=1:floor(order/2)  %pad image borders with enough for filter order
        [h,w] = size(img);
        img = [img(1,1)  img(1,:)  img(1,w);
               img(:,1)  img       img(:,w);
               img(h,1)  img(h,:)  img(h,w)];
    end
    f = gauss1d(order,sig);             %create filter coefficient matrix
    img_out = conv2(img,f,'valid');     % do the filtering
    img_out = conv2(img_out,f','valid');% do the filtering

    %---------------------------------------------------------------------------
    %   Returns coeffients for one dimensional gaussian filter.
    %   Arguments
    %           order     :  size of the mask.
    %           sig       :  scale factor.
    %   Output
    %           f          : coeffients for one dimensional gaussian filter.
    %---------------------------------------------------------------------------
    function[f] = gauss1d(order,sig)
        f = 0;i = 0;j = 0;
```

```
        % generate gaussian coefficients
        for x = -fix(order/2):1:fix(order/2)
            i = i + 1;
            f(i) = 1/2/pi*exp(-((x^2)/(2*sig^2)));
        end
        f = f / sum(sum(f)); %normalize filter

%-----------------------------------------------
%   Interface for Localization System
%   Arguments
%           No
%   Output
%           No
%-----------------------------------------------

% Initialization code - Auto generated
function varargout = building_recognition(varargin)
    gui_Singleton = 1;
    gui_State = struct('gui_Name',       mfilename, ...
                       'gui_Singleton',  gui_Singleton, ...
                       'gui_OpeningFcn', @building_recognition_OpeningFcn, ...
                       'gui_OutputFcn',  @building_recognition_OutputFcn, ...
                       'gui_LayoutFcn',  [] , ...
                       'gui_Callback',   []);
    if nargin && ischar(varargin{1})
        gui_State.gui_Callback = str2func(varargin{1});
    end

    if nargout
        [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
    else
        gui_mainfcn(gui_State, varargin{:});
    end

%-----------------------------------------------------------------------------
% Executes just before building_recognition is made visible. - Auto generated
% This function has no output args, see OutputFcn.
%   Arguments
```

```
%       hObject     handle to figure
%       eventdata   reserved - to be defined in a future version of MATLAB
%       handles     structure with handles and user data (see GUIDATA)
%       varargin    command line arguments to building_recognition (see VARARGIN)
%   Output
%       No
%---------------------------------------------------------------------------
function building_recognition_OpeningFcn(hObject, eventdata, handles, varargin)
    % Choose default command line output for building_recognition
    handles.output = hObject;
    % Update handles structure
    guidata(hObject, handles);

%---------------------------------------------------------------------------
% Outputs from this function are returned to the command line. - Auto generated
%   Arguments
%       varargout   cell array for returning output args (see VARARGOUT);
%       hObject     handle to figure
%       eventdata   reserved - to be defined in a future version of MATLAB
%       handles     structure with handles and user data (see GUIDATA)
%   Output
%       Outputs from this function are returned to the command line.
%---------------------------------------------------------------------------
function varargout = building_recognition_OutputFcn(hObject, eventdata, handles)
    % Get default command line output from handles structure
    varargout{1} = handles.output;
    set(handles.mnuBuildingMatch,'Enable','off');
    set(handles.mnuLearn,'Enable','off');
    set(handles.mnuLearnDir,'Enable','off');
    set(handles.mnuCalAngle,'Enable','off');
    set(handles.mnuALearn,'Enable','off');
    set(handles.mnuALearnDir,'Enable','off');
    set(handles.mnuMatch,'Enable','off');
    set(handles.learnPanel,'Visible','off');
    set(handles.matchPanel,'Visible','off');

% --------------------------------------------------------------------------
% hObject    handle to mnuFile (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
function mnuFile_Callback(hObject, eventdata, handles)

% --------------------------------------------------------------------
% Call back for menu Learn
%   Arguments
%       hObject    handle to mnuLearn (see GCBO)
%       eventdata  reserved - to be defined in a future version of MATLAB
%       handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
function mnuLearn_Callback(hObject, eventdata, handles)
    % Refresh display
    refresh_learn(handles)
    % Read file name
    [imgfilename pathname] = uigetfile('*.jpg','File Name');
    if pathname(1,1) == 0
       errordlg('Please select image file','Error!!!','modal');
       return
    end
    filename = [pathname imgfilename];
    set(handles.matchPanel,'Visible','off');
    set(handles.learnPanel,'Visible','on');
    set(handles.progressLbl,'String','');
    set(handles.graphlbl1,'String','Color Histogram');
    set(handles.graphlbl2,'String','Feature Points');
    disp(handles,['Learning Image ' filename]);
    % Learn image
    status = learnImage(handles,filename);
    if status == 1
        disp(handles,'Learn complete....');
    else
        disp(handles,'Learn aborted....');
    end

%--------------------------------------------------------------------
% Learn input image. Construct colour histogram and locate feature points
```

```
%   Arguments
%       handles    structure with handles and user data (see GUIDATA).
%       filename   file to be learnt
%--------------------------------------------------------------------------
function [status] = learnImage(handles,filename)
    % display input image
    axes(handles.inputImg);
    image(imread(filename));
    set(handles.inputImg,'Visible','off','Units','pixels');
    pause(0.00001);

    % Read building number
    bno = inputdlg('Building Number','Set Building Number',1);
    if isempty(bno) == 1
        errordlg('Please enter Valid Building Number','Error!!!','modal');
        status = 0;
       return
    else
        tempbno = str2num(bno{1});
        if isempty(tempbno) == 1
            errordlg('Please enter Valid Building Number','Error!!!','modal');
            status = 0;
            return
        end
    end
    bldno = int32(tempbno);

    % Locate features of the image
    tic
    [okflag,hst,shapearray,fparray,fpimg,nlblct,nlbl] = features(filename);
    tm1 = toc;
    disp(handles,['Feature point localization... Time Required ' num2str(tm1) '
seconds']);
    if okflag == 2
        disp(handles,['Given image is not colour image']);
        return;
    end
```

```
    % Display colour histogram
    axes(handles.histogramImg);
    bar(1:32,hst);colormap(gray);
    set(handles.inputImg,'Visible','off','Units','pixels');
    pause(0.00001);

    % Display local feature points
    axes(handles.featureImg);
    imagesc(uint8(fpimg));
    set(handles.featureImg,'Visible','off','Units','pixels');
    pause(0.00001);

    % Store data in files
    disp(handles,'Storing data.....');

store_features(handles.databasePath,filename,bldno,hst,shapearray,fparray,nlbl,nlb
lct)
    status = 1;

%------------------------------------------------------------------------
% Call back for menu Learn Directory
%   Argumrnts
%       hObject    handle to mnuLearnDir (see GCBO)
%       eventdata  reserved - to be defined in a future version of MATLAB
%       handles    structure with handles and user data (see GUIDATA)
%------------------------------------------------------------------------
function mnuLearnDir_Callback(hObject, eventdata, handles)
    % Read directory name
    [pathname] = uigetdir('Database Path');
    if pathname(1,1) == 0
       errordlg('Please select directory','Error!!!','modal');
       return
    end

    f_names = ls(pathname);
    noofimages = size(f_names,1);
    set(handles.matchPanel,'Visible','off');
    set(handles.learnPanel,'Visible','on');
```

```
    set(handles.graphlbl1,'String','Color Histogram');
    set(handles.graphlbl2,'String','Feature Points');

    % Learn images one by one.
    ct = 0;
    for i=1:noofimages
        l = size(f_names,2);
        if  l > 4 && (( strcmp(f_names(i,l-3:l),'.jpg')) || ( strcmp(f_names(i,l-
3:l),'.JPG')))
            ct = ct + 1;
            refresh_learn(handles);
            set(handles.progressLbl,'String','');
            disp(handles,['Learning Image ' f_names(i,:)]);
            % Learn image
            status = learnImage(handles,[pathname '\' f_names(i,:)]);
            if status == 1
                disp(handles,'Learn complete....');
            else
                disp(handles,'Learn aborted....');
            end
        end
    end
    disp(handles,['Total no of images learned : ' num2str(ct)]);

%-----------------------------------------------------------------------------
% Call back for menu Set Database Path
%   Arguments
%       hObject    handle to mnuDatabasePath (see GCBO)
%       eventdata  reserved - to be defined in a future version of MATLAB
%       handles    structure with handles and user data (see GUIDATA)
%-----------------------------------------------------------------------------
function mnuDatabasePath_Callback(hObject, eventdata, handles)
    % read database path
    [pathname] = uigetdir('Database Path');
    if pathname(1,1) == 0
        errordlg('Please select database folder','Error!!!','modal');
        set(handles.databasepathTxt,'ForegroundColor',[1 0 0]);
        set(handles.databasepathTxt,'String','Invalid database folder');
```

```
        return
    end
    % search for db_image_names.mat and db_image_des.mat at specified path
    f_names = ls(pathname);
    noofimages = size(f_names,1);
    flag = 0;
    if size(f_names,2) >=21
        for i=1:noofimages
            if ( strcmp(strtrim(f_names(i,:)),'db_image_names.dat')) || (
strcmp(strtrim(f_names(i,:)),'db_image_des')) || (
strcmp(strtrim(f_names(i,:)),'db_image_names_ac.dat'))
                flag = flag + 1;
            end
        end
    end
    if flag ~= 3
        % path specified is invalid
        errordlg('Invalid database folder','Error!!!','modal');
        set(handles.databasepathTxt,'ForegroundColor',[1 0 0]);
        set(handles.databasepathTxt,'String','Invalid database folder');
    else
        % path specified is valid. Store databse path.
        handles.databasePath = pathname;
        guidata(hObject, handles);
        % display database path.
        set(handles.databasepathTxt,'ForegroundColor',[0 0 145/255]);
        set(handles.databasepathTxt,'String',pathname);

        % turn on menu options

        set(handles.mnuMatch,'Enable','on');
        set(handles.mnuBuildingMatch,'Enable','on');
        set(handles.mnuLearn,'Enable','on');
        set(handles.mnuLearnDir,'Enable','on');
        set(handles.mnuCalAngle,'Enable','on');
        set(handles.mnuALearn,'Enable','on');
        set(handles.mnuALearnDir,'Enable','on');
        msgbox('Database path is set','Message','modal');
```

```
    end


%------------------------------------------------------------------------
% To display progress of learning image in progressLbl
%   Arguments
%       handles    structure with handles and user data (see GUIDATA)
%       dispstr    string to be displayed
%------------------------------------------------------------------------
function disp(handles,dispstr)
    str = get(handles.progressLbl,'String');
    l = size(str,1);
    str{l+1} = dispstr;
    set(handles.progressLbl,'String',str);
    pause(0.00001);

%------------------------------------------------------------------------
% To display progress of matching image in messageLbl
%   Arguments
%       handles    structure with handles and user data (see GUIDATA)
%       dispstr    string to be displayed
%------------------------------------------------------------------------
function dispmatchmsg(handles,dispstr)
    str = get(handles.messageLbl,'String');
    l = size(str,1);
    str{l+1} = dispstr;
    set(handles.messageLbl,'String',str);
    pause(0.00001);

%------------------------------------------------------------------------
% To refresh match panel.
%   Arguments
%       handles    structure with handles and user data (see GUIDATA)s
%------------------------------------------------------------------------
function refresh_match(handles)
    axes(handles.queryImg);
    image(100);colormap(gray);
    axes(handles.bestmatchImg);
```

```
    image(100);colormap(gray);
    axes(handles.viewangleImg);
    image(100);colormap(gray);
    for i = 1 : 20
        axes(handles.closeImg(i));
        image(100);colormap(gray);
        set(handles.closeImg(i),'Visible','off','Units','pixels');
        pause(0.00001);
    end

%----------------------------------------------------------------------------
% To refresh learn panel.
%   Arguments
%       handles    structure with handles and user data (see GUIDATA)s
%----------------------------------------------------------------------------
function refresh_learn(handles)
    axes(handles.inputImg);
    image(100);colormap(gray);
    axes(handles.histogramImg);
    image(100);colormap(gray);
    axes(handles.featureImg);
    image(100);colormap(gray);

% ----------------------------------------------------------------------
function mnuBuildingMatch_Callback(hObject, eventdata, handles)
% hObject    handle to mnuBuildingMatch (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% ---------------------------------------------------------------------------
% Call back for menu Angle Learn
%   Arguments
%       hObject    handle to mnuLearn (see GCBO)
%       eventdata  reserved - to be defined in a future version of MATLAB
%       handles    structure with handles and user data (see GUIDATA)
% ---------------------------------------------------------------------------
function mnuALearn_Callback(hObject, eventdata, handles)
    % Read file name
```

```
    [imgfilename pathname] = uigetfile('*.jpg','File Name');
    if pathname(1,1) == 0
       errordlg('Please select image file','Error!!!','modal');
       return
    end
    filename = [pathname imgfilename];
    set(handles.matchPanel,'Visible','off');
    set(handles.learnPanel,'Visible','on');
    set(handles.progressLbl,'String','');
    set(handles.graphlbl1,'String','Feature Points');
    set(handles.graphlbl2,'String','');
    disp(handles,['Learning Image ' filename]);

    % Learn image
    status = learnAImage(handles,filename);
    if status == 1
        disp(handles,'Learn complete....');
    else
        disp(handles,'Learn aborted....');
    end

% --------------------------------------------------------------------
function mnuCalAngle_Callback(hObject, eventdata, handles)
% hObject    handle to mnuCalAngle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


%--------------------------------------------------------------------------
% Call back for menu Learn Directory
%   Argumrnts
%       hObject    handle to mnuLearnDir (see GCBO)
%       eventdata  reserved - to be defined in a future version of MATLAB
%       handles    structure with handles and user data (see GUIDATA)
%--------------------------------------------------------------------------
function mnuALearnDir_Callback(hObject, eventdata, handles)
    % Read directory name
    [pathname] = uigetdir('Database Path');
```

```
    if pathname(1,1) == 0
       errordlg('Please select directory','Error!!!','modal');
       return
    end

    f_names = ls(pathname);
    noofimages = size(f_names,1);
    set(handles.matchPanel,'Visible','off');
    set(handles.learnPanel,'Visible','on');
    set(handles.graphlbl1,'String','Feature Points');
    set(handles.graphlbl2,'String','');

    ct = 0;
    for i=1:noofimages
        l = size(f_names,2);
        if  l > 4 && (( strcmp(f_names(i,l-3:l),'.jpg')) || ( strcmp(f_names(i,l-
3:l),'.JPG')))
            set(handles.progressLbl,'String','');
            disp(handles,['Learning Image ' f_names(i,:)]);
            % Learn image
            status = learnAImage(handles,[pathname '\' f_names(i,:)]);
            if status == 1
                disp(handles,'Learn complete....');
            else
                disp(handles,'Learn aborted....');
            end
            ct = ct + 1;
        end
    end
    disp(handles,['Total no of images learned : ' num2str(ct)]);

%-------------------------------------------------------------------------------
% Learn input image for angle calculation. Locate feature points
%   Arguments
%       handles    structure with handles and user data (see GUIDATA)
%       filename   file to be learnt
%-------------------------------------------------------------------------------
function [status] = learnAImage(handles,filename,bldno)
```

```
% Display input image
axes(handles.inputImg);
image(imread(filename));
set(handles.inputImg,'Visible','off','Units','pixels');
pause(0.00001);

% Read building number
bno = inputdlg('Building Number','Set Building Number',1);
if isempty(bno) == 1
    errordlg('Please enter Valid Building Number','Error!!!','modal');
    status = 0;
   return
else
    tempbno = str2num(bno{1});
    if isempty(tempbno) == 1
        errordlg('Please enter Valid Building Number','Error!!!','modal');
        status = 0;
        return
    end
end
bldno = int32(tempbno);

% Locate feature points of the image ( Harris detector )
tic
[fpimg,fparray] =  Harris_detector(filename);
toc

% Display feature points
axes(handles.histogramImg);
imagesc(uint8(fpimg));colormap(gray);
set(handles.featureImg,'Visible','off','Units','pixels');
pause(0.00001);

% Store data in files
store_features_angle(handles.databasePath,filename,bldno,fparray);
status = 1;

%#########################################################################
```

```
function matchImg(handles,filename)

    set(handles.learnPanel,'Visible','off');
    set(handles.matchPanel,'Visible','off');
    set(handles.messageLbl,'String','');
    pause(0.00001);
    set(handles.matchPanel,'Visible','on');

    set(handles.queryImg,'Visible','off');
    set(handles.bestmatchImg,'Visible','off');
    set(handles.viewangleImg,'Visible','off');

    refresh_match(handles);

    % Display query image
    axes(handles.queryImg);
    image(imread(filename));
    set(handles.queryImg,'Visible','off','Units','pixels');
    pause(0.00001);

    % ---------------------------------------------------------------------
    tic
    % Stage1 step 1
    [okflag,qryhst,qryshapearray,qryfparray,qryfpimg,qrynlblct,qrynlbl] =
features(filename);
    if okflag == 2
        disp(handles,['Given image is not colour image']);
        return;
    end

    % Match color descriptor with database images.
    [stage1img,stage1bno] = match_huedes(handles.databasePath,qryhst);

    dispmatchmsg(handles,['Query Image : ' filename]);
    dispmatchmsg(handles,'Stage 1 : step 1.....');

    % ---------------------------------------------------------------------
    % Stage1 step 2
```

```
    % Match shapes with database images.
    dispmatchmsg(handles,'Stage 1 : step 2.....');
    [stage2img] =
match_shapes(handles.databasePath,stage1img,stage1bno,qryshapearray);
    dispmatchmsg(handles,'Stage 1 step 2 output.....');
    % Display images selected in stage2
    no_of_stage2img = size(stage2img,2);
    dispmatchmsg(handles,['No of images : ' num2str(no_of_stage2img)]);
    for i = 1 : no_of_stage2img
        set(handles.closeImg(21-i),'HandleVisibility','on');
        axes(handles.closeImg(21-i));
        dbfilename = stage2img{i};
        image(imread([handles.databasePath '\' dbfilename(1:size(dbfilename,2)-3)
'jpg']));
        set(handles.closeImg(21-i),'Visible','off','Units','pixels');
        pause(0.00001);
    end

    % -------------------------------------------------------------------
    % Stage1 step 3
    dispmatchmsg(handles,'Stage 1 step 3.....');
    tic;
    [bestmatch, bestmatch_bno] =
match_local_features(handles.databasePath,stage2img,qryfparray,qrynlblct,qrynlbl);
    tm1 = toc;
    dispmatchmsg(handles,['Best match... Time Required ' num2str(tm1) '
seconds']);

    % Display best match
    axes(handles.bestmatchImg);
    image(imread([handles.databasePath '\' bestmatch(1:size(bestmatch,2)-3)
'jpg']));
    set(handles.bestmatchImg,'Visible','off','Units','pixels');
    pause(0.00001);

    % -------------------------------------------------------------------
    % Stage2
    tic;
```

```
    dispmatchmsg(handles,'Stage 2.....');
    % Locate feature points of the query image
    [qryfpimg,qryfparray] =  Harris_detector(filename);
    [viewangle] =
match_hcfp(handles.databasePath,qryfparray,qryfpimg,bestmatch_bno);
    tm2 = toc;
    dispmatchmsg(handles,['View Angle... Time Required ' num2str(tm2) '
seconds']);

    dispmatchmsg(handles,['Total Time Required : ' num2str(tm1 + tm2) '
seconds']);

    % Display view angle image.
    axes(handles.viewangleImg);
    image(imread([handles.databasePath '\' viewangle(1:size(viewangle,2)-3)
'jpg']));
    set(handles.viewangleImg,'Visible','off','Units','pixels');
    pause(0.00001);

% --------------------------------------------------------------------
% Call back for menu Match
%   Arguments
%       hObject    handle to mnuMatch (see GCBO)
%       eventdata  reserved - to be defined in a future version of MATLAB
%       handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
function mnuMatch_Callback(hObject, eventdata, handles)

    set(handles.learnPanel,'Visible','off');
    set(handles.matchPanel,'Visible','off');
    set(handles.messageLbl,'String','');
    pause(0.00001);
    set(handles.matchPanel,'Visible','on');

    set(handles.queryImg,'Visible','off');
    set(handles.bestmatchImg,'Visible','off');
    set(handles.viewangleImg,'Visible','off');
```

```
    % Refresh display.
    refresh_match(handles);

    % Read query image path
    [imgfilename pathname] = uigetfile('*.jpg','File Name');
    if pathname(1,1) == 0
       errordlg('Please select image file','Error!!!','modal');
       return
    end
    filename = [pathname imgfilename];
    matchImg(handles,filename);

% --------------------------------------------------------------------------
% Call back for menuMatchDir
%   Arguments
%       hObject    handle to mnuMatch (see GCBO)
%       eventdata  reserved - to be defined in a future version of MATLAB
%       handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------------
function mnuMatchDir_Callback(hObject, eventdata, handles)

    % Read directory name
    [pathname] = uigetdir('Query Path');
    if pathname(1,1) == 0
       errordlg('Please select directory','Error!!!','modal');
       return
    end

    f_names = ls(pathname);
    noofimages = size(f_names,1);
    set(handles.matchPanel,'Visible','off');
    set(handles.learnPanel,'Visible','on');
    set(handles.graphlbl1,'String','Feature Points');
    set(handles.graphlbl2,'String','');

    ct = 0;
    for i=1:noofimages
        l = size(f_names,2);
```

```
        if  l > 4 && (( strcmp(f_names(i,l-3:l),'.jpg')) || ( strcmp(f_names(i,
3:l),'.JPG')))
            refresh_match(handles);
            set(handles.progressLbl,'String','');
            matchImg(handles,[pathname '\' f_names(i,:)]);
            ct = ct + 1;
            pause(4);
        end
    end
    dispmatchmsg(handles,['Total no of images queried : ' num2str(ct)]);
```