Viewpoint Invariant Content Based Image Retrieval for Urban Navigation

Gareth Gwynn
University of Bath
MMath

2005
Viewpoint Invariant Content Based Image Retrieval for Urban Navigation

Submitted by Gareth Gwynn

COPYRIGHT
Attention is drawn to the fact that copyright of this thesis rests with its author. The
Intellectual Property Rights of the products produced as part of the project belong to the
University of Bath (see http://www.bath.ac.uk/ordinances/#intelprop).
This copy of the thesis has been supplied on condition that anyone who consults it is
understood to recognise that its copyright rests with its author and that no quotation from
the thesis and no information derived from it may be published without the prior written
consent of the author.

Declaration
This dissertation is submitted to the University of Bath in accordance with the
requirements of the degree of Bachelor of Science in the Department of Computer
Science. No portion of the work in this dissertation has been submitted in support of an
application for any other degree or qualification of this or any other university or
institution of learning. Except where specifically acknowledged, it is the work of the
author.

Signed . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

This thesis may be made available for consultation within the University Library and may
be photocopied or lent to other libraries for the purposes of consultation.

Signed . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Abstract

This document follows the research and development of a system which identifies buildings on the University of Bath campus from a 640x480 photograph taken on a standard Mobile-Camera Phone. The system relies on feature detection techniques and the comparison of colours in the image to compare the query image to a database. The system assumes the buildings in question to be planar in order to compute a homography between the database and test images – the quality of which is tested by a variety of techniques. The final system performs well at viewpoint invariance and shows evidence of good performance in poor resolution – however, implementation of the system in Matlab 6 leads to a greater need for a method of hashing the database and the final results highlight an accuracy verses efficiency trade-off.
Acknowledgements

I would like to thank my project supervisor Dr. John Collomosse for his help and guidance in the making of this project and for putting up with some quite unfeasibly long emails on the progress of the project. I would also like to thank the BUCS staff, particularly Tom Natt for assistance in all manner of computer based enquiries and Chris Day and Lyndsay Fenner for the lending of a digital camera for the photographs and for filling in the Computer-Science gaps left from doing a Mathematics degree.
# Contents

1. Introduction ........................................... 1

2. Literature Review ..................................... 2
   2.1 Introduction ...................................... 2
   2.2 Content Based Image Retrieval .................. 2
   2.3 Multiview Geometry .............................. 5
   2.4 Mosaicing and Matching ......................... 7
   2.5 Illumination ..................................... 12
   2.6 Summary ......................................... 12

3. Software Processes and Requirements ................. 15
   3.1 Introduction ..................................... 15
   3.2 Defining System Boundaries .................... 16
   3.3 Identifying and Prioritising Requirements .... 16
   3.4 Testing and Evaluation .......................... 19
   3.5 Resources ....................................... 20

4. Design of Core System ................................ 21
   4.1 Architecture Design .............................. 21
   4.2 Implementation of Basic System ............... 22
   4.3 Testing .......................................... 29

5. Refining the System ................................... 31
   5.1 Illumination Invariance ......................... 31
   5.2 Edges ............................................ 34
   5.3 Corners .......................................... 38
   5.4 Improving the Stability of the System ....... 40
   5.5 Building a Combined System .................... 44
   5.6 Verification and Evaluation ..................... 47

6. Conclusion ............................................ 57
   6.1 The Project ...................................... 57
   6.2 The Software Process ............................ 57
   6.3 Future Work ...................................... 58

Bibliography ............................................ 59
Appendix ................................................. 63
1. Introduction

With the increasing popularity of mobile telephones with a camera facility, research is currently being done into using this technology to aid navigation – particularly in an urban environment [1]. There are, of course, non-vision based methods of determining a mobile telephone handset location – by methods such as signal triangulation. However, a vision based system would be an alternative and in practice, the two could be used together.

This would be particularly useful in areas where GPS fails to work. Errors in GPS signal can occur due to delays in the signal passing through the atmosphere, or reflecting off tall buildings when in an urban environment [2]. The relative position of the satellites is also a factor, with tighter angles causing less accurate results. Plus, signals are capable of being blocked by buildings, electrical signals or even dense foliage.

As such, the overarching aim of the project is for the user to take a tour of the University of Bath campus with a digital camera, taking photographs of various buildings and use the program developed to identify the query images. The problem is one of Content Based Image Retrieval (CBIR) and previous methods of such systems are studied in the literature review (Section 2) as well as methods of coping with viewpoint invariance, an important aspect to the success of the project.

The choice of the campus as the urban environment is something of an arbitrary one owing mainly to its convenience. A map of the campus is seen in Appendix A, and Appendix B contains the images which are eventually used as the database for the program (implemented in section 5). The quality of the images is determined by the problem description. The higher quality camera phone images tend to be 640 pixels by 480 pixels so for this reason, this has been chosen as the standard resolution throughout the program, although to reflect the poorer quality images such devices are capable of taking, the testing section includes studies into poorer resolution.

The query image is to be compared to the catalogue of known images until the best match is made. The viewpoint invariance referred to in the project title stems from the idea that the person taking the query image is unlikely to be standing in the identical position to the person who constructed the catalogue. In addition to viewpoint invariance, such a system would need to be robust to noise and clutter in the image and resistance to poor illumination will also be looked into thoroughly.

As will become clear throughout the document, following the literature review an evolutionary design model is adopted. As Such, section 4 describes the construction of a very basic system capable of the transform and comparison of planar objects, while section 5 describes how this system was adapted and advanced in order to make a fully working model system. Throughout the project, several themes are studied, the most clear being the accuracy/efficiency trade-off which underlines most of the second half of the report and the success of this is assessed via a test set which encompasses images of poor quality, at extreme angles and in poor illumination.
2. Literature Review

In this section I shall give a brief overview of the history of this problem domain as well as the work done in the various fields relating to the project, their origins and their relevance to what the project is trying to achieve.

2.1 Introduction

The key aim of the project is to construct a system capable of identifying the facades of buildings on the University Campus based on a catalogue of images. The problem is primarily one of Content Based Image Retrieval (CBIR) – in particular, I have chosen to approach it as a problem of viewpoint invariant CBIR of planar objects [2]. The choice of planar objects at this early stage is an important simplifying constraint, increasing the number of easily implemented methods. A query image will have to be compared to a catalogue of known images until the best match is made. The viewpoint invariance stems from the idea that the person taking the query image is unlikely to be standing in the identical position to the person who has constructed the catalogue.

The purpose of this literature review is to look into the relevant areas of computer vision which I will need to use in order to construct such a system and analyse previous attempts at solving similar problems. The main areas which this review will consider, aside from CBIR, are methods of dealing with Multi-view Geometry and mosaicing. In addition to this, methods of colour correction and dealing with varying illumination will be considered.

2.2 Content Based Image Retrieval

The phrase “Content Based Image Retrieval” was first used by Kato [22] in 1992 to describe a system which retrieved images from a database based on colour and shape. The term is used to describe the retrieval of images from a large collection on the basis of features that can be extracted from the images themselves, rather than from, say associated keywords or file names [7]. As large collections of images stored on computers became commonplace, the need for a system capable of categorising images semantically increased. Consequently, users will require a method with which to search and browse through images effectively.

Content Based Image Retrieval has been of increasing interest as the field of Computer Vision has expanded, and in 1999, Eakins and Graham prepared a report for the Joint Informations Systems Committee (JISC) Technology Applications Programme with the intention of reviewing the current capabilities (or limitations) of CBIR systems which were then available [7]. At the time, methods of searching through a database of images were almost entirely limited to text based enquiries which were split into various levels - ranging from “Level 1” which requested very basic shapes or colours, to “Level 3” which described specific scenes (such as “Folk Dancing”). The report states that, at the time of writing, CBIR systems were limited almost entirely to Level 1 applications – making use of the most primitive features such as colour, texture and shape.

Typically, the user would search by name with a level 1 enquiry searching by colour, shape or texture (with colour and texture being the more reliable methods in systems created so far, as shape suffers from foreground/background problems). A more detailed look at illumination and
colour spaces will follow in Section 2.5. An effective way of searching by colour is to use analysis by comparison of Colour Histograms [7]. For each image in the database, a colour histogram is constructed, which simply displays the proportion of each colour in the image. Users can search by colour (i.e. using RGB colour sliders) or alternatively, can submit a query image, whose colour histogram can be compared to those contained in the database – with the closest matches being returned. In the case of this particular project, the latter is the more useful method.

The matching technique most commonly used, “histogram intersection”, was first developed by Swain and Ballard [23] in 1991, which discretised the colours in the image in order to draw a 3-dimensional histogram which could be compared to others. Colour, as a means of identification, had been somewhat neglected until this paper, which claimed the method was robust to background distraction, viewpoint invariance, occlusion and resolution. The histograms created are invariant to translation and rotation and change only slowly with angle, scale and volume.

With particular reference to application within this project, disadvantages in this system arise with issues of illumination, so in order for the system to be investigated, suitable adjustments will have to be made. Depending on the error in such a system, it is unlikely that the same building will return identical colour histograms unless the ambient lighting is identical. In low light conditions, particularly, this method is likely to become unreliable – something that will be discussed in more detail in section 3.

Texture, as a means to matching images, is particularly useful in cases such as this, as it can be used to identify features such as sky or grass. Matching textures is generally done using pairs of pixels and analysing their relative brightness. Changes in contrast and direction have been studied and it is possible in some systems for the user to query texture as they would colour. Work has also been done to let users submit query images (i.e. a photograph of a sample of grass) – something particularly relevant to this project [7].

There are several ways to approach the texture problem. One is to use Fourier methods – monitoring the horizontal and vertical frequencies and group this data in a way so as to gain a set of measurements for the texture [3]. Alternatively (or in conjunction with this) we can apply statistical methods. The first, and still one of the most popular of these methods is the co-occurrence matrix method, which has been in use since 1973. The matrix contains pixel pairs for specific brightness levels when separated by a specific distance and inclination. It is unlikely that the project will need to consider texture to this degree, especially at the suggested resolution of 640x480 where such detail is likely to be lost.

Our own ability to recognise objects is believed to be strongly reliant on the shape of the object in question [24] and shape is a well defined concept [7]. The major hurdle with object recognition is geometric invariance [25, 7]. Early attempts like template matching [26] (which performed every possible transformation on one image and moved the template pixel by pixel over the image) became impractical as the range of possible transforms became too great (rotation, projection, etc.).
Weiss [25] developed a system where so called “geometric invariants” are determined by the geometry of the shapes. Such descriptors remain unchanged under geometric transformation (for example, changing the viewpoint). Finding projective invariants (quantities that are unchanged under projective transform) are particularly useful, as we shall see in section 1.3, the transforms used on the query image are likely to be projective.

The problem with CBIR of this nature is that, without relying on metadata, the photographs themselves contain no information in the way that say, a report does. A written document can be searched for by text based fields as varied as author, keywords, page number and so on. [17].

At the time, object recognition by feature analysis (which is the overall aim of the project) was described as a “grey area” between image processing and CBIR. There was, however, some work into developing a system that would let the user submit a simple drawing as a search (the example used was a sketch of a typical fish) by Jacobs et al [31]. Such systems would fall under the “Level 2 category” [7]. A logical extension of this is Query by Pictorial Example (QPE), introduced by Chang and Fu [27] which lets the user enter a sample image and the system searches for examples of a similar kind.

A procedure to classify a query image is shown diagrammatically in Figure 1. The presentation [18] was discussing ways to classify images into one of several pre-defined classes (using what we previously referred to as level 1 features of colour, texture and position). The purpose of such classification was not difficult to imagine. Attempting to retrieve an image using a query image is far easier, and efficient, computationally, if the query image is searched for features first. Then comparisons are restricted to the groups of images with relevant features.

This sorting of the catalogue into groups of images is “indexing” and is being developed for, amongst other things, medical purposes [19]. In the case of this particular project, most level 1 features will be very similar (buildings tend to be square, with square windows and so on) but perhaps splitting the database by colour, or even number of windows on the front of the buildings will have an effect on the speed of the comparison algorithm.

The alternative to this is to use a process known as “Hashing” the database. This method assigns various “keys” (numbers representing each of the images) to locations (which would categorise various features). Hashing is rarely a one-to-one relationship and in the case of, say, window numbers, there could be several images in the database with the same number of windows [20] though it could efficiently limit the search space.

![Figure 2.1: Procedure to classify a query image into one of several pre defined classes by feature detection.](image)
In the introduction, a paper by Robertson and Cipolla [1] was referred to, implementing a system similar to the one being researched. The paper describes how, following the use of a Harris corner detector to identify interest points (which I shall come to later), we obtain “local image regions” by sampling 8x8 windows of pixels. The method is particularly geared towards “multi-scale” matching, and relies on Dumfornaud et al.’s methods of matching images with different resolutions [21].

2.3 Multi-view Geometry

Before we can start comparing our query image with images from the database in any kind of detail, we will need to take account for the viewpoint invariance of the problem. The user can take the photograph of the building from a variety of angles. In order to be able to extract the correct information from the query image, we will need to consider Multiview Geometry.

There are two systems we can use for our images - perspective or affine projection. A perspective camera uses the mathematics of similar triangles (see figure 2.2) and considers light, travelling in straight lines, leaving the real world and travelling to a focus point. The light passes through a “window”, making an image of the real world on the window. Using this system, we divide each real world point by the z coordinate, effectively losing the concept of depth, and transferring a 3-dimensional system into into a 2-dimensional system, yet retaining the concept of perspective [10]. In matrix form the perspective transform matrix is

\[
P = \begin{pmatrix}
    d & 0 & 0 & 0 \\
    0 & d & 0 & 0 \\
    0 & 0 & d & 0 \\
    0 & 0 & 1 & 0
\end{pmatrix}
\]

which implies the resultant points on the window \((x',y',z',1)^T\)

So in order to transform a 3-dimensional world point in homogeneous coordinates onto a single plane \((x,y,z,1)^T\), we multiply by the matrix...

\[
P \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} dx \\ dy \\ dz \\ 1 \end{bmatrix}
\]

Which implies the resultant points on the window \((x',y',z',1)^T\)
\[
x' = \frac{dx}{z} \\
y' = \frac{dy}{z} \\
z' = d
\]

This single plane which all the points have mapped to is the image or camera plane.

An alternative is an affine camera which assumes parallel light rays, with points in the 3-dimensional world projected straight onto a 2-dimensional coordinate system. Affine cameras preserve length of vectors parallel to the window, since the rays of light are parallel to the plane.

For their implementation of an imaged based system, Robertson and Cipolla [1] used perspective projection and a wide baseline matching algorithm. In order to understand wide baseline matching, we need to be familiar with epipolar geometry - the geometry of two cameras.

The epipolar plane is defined by the foci of the two cameras, \( f_1 \) and \( f_2 \) and the object we are interested in \( X \) (which is a single point in our 3-dimensional scene). Associated with each of our cameras is a window plane, which acts as our image of our 3-dimensional world. The line between the two foci is known as the baseline. Where the line \( f_1X \) intersects the window plane is where that point is represented in the image, \( y_1 \) (figure 2.3).

The object \( X \) is also represented in the second window plane – but even if we were unsure of the objects actual position, we could still determine possible locations for it in the second image, from our knowledge of the first image. Based on the information on the first image, \( X \) could lie anywhere on the straight line passing through \( f_1y_1 \). Being a point, its distance from the plane makes no difference to its perceived size.

This means that the line \( f_2X \) could intersect the line passing through \( f_1y_1 \) at any point beyond \( y_1 \), implying that possible locations for \( y_2 \) (the point of intersection of \( f_2X \) with the second image plane) must lie on the line intersecting the plane and the window (figure 3) [5, 10].

![Figure 2.3: Features of Epipolar Geometry and the possible positions of \( y_2 \) in the second window plane](image)
This line is known as the epipolar line and its existence gives rise to the “epipolar constraint”. Using the above description, we can derive the “Fundamental Matrix”. This matrix describes the epipolar geometry, in particular, the point-to-line mapping.

A less general mapping is a Homography. Homographies are projective transforms [6] that map one plane to another. This idea of using a homography to relate two images is obviously important for our attempts to identify planar objects from two different viewpoints. In the next section, we will look at the how we can derive a specific homography between two images with potentially overlapping content.

2.4 Mosaicing and Matching

The aim of the project is to match a query image with an image from our collection. In the paper by Cipolla and Robertson [1], a method that matched features in the two images using RANSAC [12] was used. In this section, I will review the RANSAC method, as well as a process called Image Mosaicing, which relies on the overlapping of images, and discuss how this could be used to identify images in an urban environment.

Mosaicing is the stitching together of one or more images to make a larger image. Attempts to make a panorama from a series of photographs tend to fail as it is difficult to get the pictures to overlap successfully due to the image plane being tilted as the camera is manoeuvred into position for the second photograph. To transform the image successfully, a projective transform is required [5] rather than merely relying on translations, rotations and scale [8, 10].

The concept of mosaicing is based on the images having an overlap in their content, allowing us to compute a homography between the two so as to put them into the same image plane. We then apply this homography to one of the images so the two are then suitable for stitching together seamlessly [10, 8, 6].

In order for mosaicing to be successful, there will need to be an overlap between the images. The application to this particular project is that, given a query image and a selection of candidate images from the database, we can compute homographies between query and database images in attempting to mosaic the two. We can then assess the quality of the results and if mosaicing is successful, then the likelihood of a match is greater.

Before we begin looking at how we match points for mosaicing techniques, we need to look into ways of feature detection to begin mapping.

2.4.1 Feature detection

The most basic type of feature detection is low–level, defined by Nixon and Aguado [3] as those basic features that can be extracted automatically from an image without any shape information (i.e. information about spatial relationships).

The most basic of these features are edges – the great advantage with these being that such detection tends to be insensitive to the overall level of
illumination [3], increasing its versatility. Edge detection relies on detecting contrast in the image with the points of greatest change (i.e. those with the greatest gradient) which are the edges. As we run across a “scan line” on an image, we can see where the edges are. Differentiating horizontally across the image will detect the vertical changes, and vice versa [3]. However, for our purposes, edges are not accurate enough. We are specifically interested in distinguishable points, present in both our images [9] - the query and the potential matching database image. In order to compute a single possible homography, four points are required [6]. A corner detector is advantageous, not only because corners are preferable over edges (being points, rather than lines), but because corners are viewpoint invariant.

There are several different corner detectors – however, perhaps the most reliable for our purpose is the “Harris Corner Detector”, developed by Harris and Stevens in 1988 [11]. The procedure is to greyscale the image and then differentiate horizontally (by applying a 3x3 window in a convolution with the original image, implemented in [9, 10 and 3]). Another 3x3 window allows us to differentiate vertically.

If I is our image, by doing this we are producing fields for both dl/dx and dl/dy and combining them so as to pick out points subject to sharp increases in both the x and y directions. We combine the two fields by a matrix function M (equations 4 and 5 [9]), the local maxima of which correspond to image corners [9].

\[
M = \begin{pmatrix}
\left(\frac{\partial I}{\partial x}\right)^2 & \left(\frac{\partial I}{\partial y}\right) \\
\left(\frac{\partial I}{\partial x}\right) & \left(\frac{\partial I}{\partial y}\right)^2
\end{pmatrix}
\]

\[
R = \det M - k (\text{trace}M)^2
\]

If our query and database images do correspond, feature detecting both images should reveal a series of points common to both images. The points we seek are known as “invariance properties”, as the process we use to extract the images should ideally be immune to factors such as illumination.

The next step is to match such points across the two images in order to compute the required homography. If the difference in focal point between the two images is small, then we take our point in image A and only search a small region around the same point in image B until we find our corresponding image point.

However, in this project, we are unlikely to be that fortunate and our two images may be taken from significantly different positions (i.e. a wide baseline – see section 1.3). A much better method of matching points between images in a situation such as this is RANSAC.
2.4.2 RANSAC

Random Sample Consensus (RANSAC) is a robust estimation technique and was introduced by Fischler and Bolles [12]. It is robust to “outliers” (incorrect corner correspondences) and is a very effective algorithm for matching points across images. The original paper set out to solve what had become known as “The Location Determination Problem” (LDP) which set out to find the location from which an image was taken, given the 3-dimensional co-ordinates of “m control points” (given subset of the m control points is visible in the image). The idea of finding the geometric relationship between two different images of the same object is what interests us, and below, I have outlined how we can determine the homography which relates the two images.

The RANSAC algorithm computes the optimal Homography between two images. A homography is commonly represented in the form of a 3x3 matrix, which transforms one set of homogenous 2-dimensional points to another as shown in equation 6.

\[
\begin{pmatrix}
 b_i \\
 b_i \\
 1
\end{pmatrix} = \begin{pmatrix}
 h_{11} & h_{12} & h_{13} \\
 h_{21} & h_{22} & h_{23} \\
 h_{31} & h_{32} & h_{33}
\end{pmatrix} \begin{pmatrix}
 a_i \\
 a_i \\
 1
\end{pmatrix}
\]

(6)

In equation 6, the 2x2 matrix comprising of \(h_{11}, h_{12}, h_{22}\) are responsible for shear, rotation and scale, with \(h_{31}, h_{32}\) being responsible for perspective-like distortion and \(h_{31}, h_{32}\), responsible for translations. With these, we can relate any two views of the same planar object and it is the RANSAC algorithm which determines the homography H.

The method is to select a number of pairs of points (usually four) between the two images at random. Assuming these points to correspond, we use these randomly selected points to compute a homography between the two images [6]. We do this by, for each point correspondence \(p_i = (x_i, y_i)^T \rightarrow q_i = (x'_i, y'_i)^T\) (for \(i=1, 2, 3, 4\)), we solve the following set of linear equations:

\[
(h_{11} a_{i1} - h_{12} a_{i2} - h_{13}) - b_{i1}(h_{31} a_{i1} + h_{32} a_{i2} + h_{33}) = 0
\]

(7)

\[
(h_{21} a_{i1} - h_{22} a_{i2} - h_{23}) - b_{i2}(h_{31} a_{i1} + h_{32} a_{i2} + h_{33}) = 0
\]

(8)

Equations 4 and 5 are found by multiplying out equation 3. In order to calculate the values of \(h\) in the above equation, we construct a 8x9 matrix from equations (4) and (5) for each of the four points.
We need to solve this equation, to get the values of \( h \), so will have to deal with \( A \). The smallest singular vector of \( A \) is the best in the least squares sense \([10]\) and the method for solving most linear least squares problems is Single Value Decomposition (SVD) \([32]\). SVD is based on the theorem that any matrix \( A \) can be written as the product of 3 other matrices, with specific properties. Following the determination of the homography, we usually normalise the matrix by performing the operation

\[
H = H./H(3,3) \tag{10}
\]

which makes no difference to the overall calculation – however, the fact such a normalisation is possible demonstrates that the homography has 8, not 9, degrees of freedom.

RANSAC is a search mechanism to minimise an error function and therefore find the optimal homography. The process described above determines a possible homography which must then be tested for its accuracy. After applying the homography to one of the images, we test each feature point outside of the samples for its closeness to other feature points. Before doing this, we need to define a "threshold", so we can determine whether a pair of points being tested can be defined as "close" or not.

We iterate this method, randomly selecting the 4 points that form the foundation of our homography, and recording the quality of the homography achieved (based on the number of pairs beneath the "close" threshold). After a predefined number of iterations, the best quality homography is the one that is chosen \([5]\).

There are several ways of determining the best homography.

The most straight-forward is, given 2 images (A and B) we apply the homography to all the features in A and identify the features in B which they are closest to. Then for each \( a \in A \) we define a distance operator

\[
d(a, H) = \min_{b \in B} |b - H a| \tag{11}
\]
And define a distance transform for each H:

$$D(H) = \sum_{a \in D} d(a)$$  \hspace{1cm} (12)

Finally, we select the H with the smallest measure of D(H) [10]. An alternative and superior method is to minimise the “Forward Backward transfer error”, given by:

$$E(H) = \sum_{i=1}^{4} |Hp_i - q_i| + |Hq_i - p_i|$$  \hspace{1cm} (13)

with the smallest value of E(H) being the chosen homography. Following this, we can refine this estimate further using the mean squared RGB error between image pixels which overlap when transformed by H, given by:

$$E'(H) = \frac{1}{|x|} \sum_{x \in x} |I'(Hx) - I(x)|^2$$  \hspace{1cm} (14)

where x is the subset of overlapping pixels within all the pixels in image I [28].

### 2.4.3 Finding the optimal homography

The methods discussed so far will determine the optimal homography between the query image and the image selected from our catalogue of building facades. However, this process will be run on several candidate images from the catalogue and a potential homography will be found in each case. For example, through various methods, we may have got our selection of candidate images down to say, four – however, we will have tried to compare our query image with four different database images. We now need to assess which of the four is the correct match.

With each of the four, we will have found a region in which our query and candidate image are supposed to overlap. Following a transform of one of the images, we measure the quality of that overlap. There are several ways to do this. Distance methods, such as Mean Squared Error, rely on measuring the distance between pairs of corresponding feature points in the two images and finding the average of their squared distances apart. However an alternative to measuring distance is to consider the colour of the pixels in the supposed overlap.

One such method is the principle of least-squares, which relies on comparing the colours of corresponding pixels. For the overlapping pixels in Images 1 and 2 (I₁ and I₂), the least square value is given by...

$$E = \frac{1}{N} \sum_{x=1}^{x_{\text{max}}} \sum_{y=1}^{y_{\text{max}}} \left| I_1(x, y) - I_2(x, y) \right|^2$$  \hspace{1cm} (15)

where N is the total number of pixels \((x_{\text{max}} \times y_{\text{max}})\). This gives a mathematical figure which makes the images easier to compare. This has given us a strong methodology with which to approach the design of the project. The first step is to determine the homography using features-detection based metric and compare results using a data/pixel based metric.
Problems with this arise from differences in illumination between the two images – which may produce unfavourable results for images which are actually a correct match. Cipolla and Robertson [1] suggest a pixel normalisation formula to help achieve robustness in lighting variation (described in section 2.5). We need to consider ways in which colours are “close” to one another and ways in which illumination affects colour perception.

### 2.5 Illumination

Throughout this chapter, the issue of how we can use colour to our advantage has been mentioned. CBIR systems have been seen to use colour histograms as a means of retrieving images (section 2.2) and in an implementation of a system similar to the one I intend to make, it was observed that analysing frequent colour could be used to aid elimination of unlikely database views (and was suggested as a possible extension) [1].

However, colour issues can be a hindrance too. Photographs taken at differing times of day will have different shadows and even different colours as the amount of daylight changes. Even in identical lighting conditions to those in which the catalogue image is taken, the change of angle will have an affect which can be reversed mathematically by “cosine correction” (something in-built to more expensive light meters) [13]. The change in appearance of illumination of a matt surface due to the angle at which it is viewed is called Lambertian reflection [33]. Such surfaces appear equally bright from all viewing angles because they reflect light with equal intensity in all directions and, as such, brightness only depends on the angle between the surface normal and the direction to the light source.

Work has been done to estimate the lighting situation from a given image which would, in our case, let us adjust the image accordingly to bring it in line with the images in our database. Experiments by Trussell and Vrheil in 1991 [14] obtained varied results depending on the precise method used. More recently, Rizzi, Gatta and Marini [15] have based a model closely on the human eye, allowing it to adapt in different lighting conditions and continue to extract visual information from the scene. The Cipolla and Robertson paper [1] attempts to achieve robustness to lighting variation by normalising the pixel intensity according to:

\[
\hat{I}_{uv} = \frac{I_{uv} - \bar{I}}{\sqrt{\frac{1}{N} \sum_{u,v} |I_{uv} - \bar{I}|^2}}
\]

(12)

Where \( I \) is the pixel intensity, \( \hat{I} \) is the normalised value, \( \bar{I} \) is the mean and \( N \) is the number of pixels.

The casting of shadows is another factor that will change depending on the time and the conditions the system is in use. Calculations of “Shadow Factor” [16] depend on readings of the illuminations of both the object, and the shadow itself, making such measurements impractical for implementation in this particular project.
In order for us to begin to discuss how we can alter colour, we need to consider ways in which colours are “close” to one another. A colour space is a means of uniquely specifying a colour [29]. Examples of colour spaces include HSL (also known as HSI) which stands for Hue, Saturation and Lightness (or Intensity) and HSV (Hue, Saturation and Value). Figure 4 shows a HSL colour space [30] in which hue is an angle from 0 to 360 degrees, saturation ranges from 0 to 1 (sometimes 0 to 100%) and defines how grey the colour is (0 indicates grey and 1 is the pure primary colour). Finally, lightness also varies from 0 to 1. Thinking of the discs of varying lightness piled on top of each other in order of increasing brightness makes us think of the space as a cylinder of varying colour.

An alternative to such a system is the RGB colour cube (fig 5). Each colour (red, green and blue) is assigned to each of the axis. Each colour in the cube (which we can think of as a solid) is then defined by a three dimensional co-ordinate (known as a “(r,g,b) triple”). The diagonal line from (0,0,0) to (1,1,1) scales from white to black via a line which represents all the shades of grey. However, this system does not represent every colour we can see (and is therefore said to lie “within our perceptual space”).

These ways of thinking of colour gives us a way of considering differing colours relation to each other, geometrically. We can see how the colour would change in brighter light (increase RGB), lower light (decrease RGB) or in a certain shade (increase each component accordingly). In this sense, the error given to a colour measurement can include a region in these spaces.

### 2.6 Summary

The purpose of this literature review has been to look into existing work in the field of Computer Vision that could aid the development of a system for navigation in an urban environment.

In order to identify the query image against a database of views, a number of methods will clearly have to be implemented to reduce the set of possible matches to one. Colour, texture and basic shape could be used to identify some possible matches, and eliminate the more unlikely possibilities before the more computationally expensive methods of RANSAC and mosaicing are implemented.

Figure 2.4: HSL Colour space diagrams representing increasing values of lightness ranging from 0.0 (top left) to 0.90 (bottom right) at 0.10 intervals. A lightness of 1.00 is totally white. Images taken from [30]
However, certain aspects of what has been researched will need to be tested before their success can be assessed. For example, methods of colour correction could be used on the query image right at the start of the process (prior to any CBIR), or could be implemented prior to the testing of mosaic quality nearer the end of the chosen method.

The only way to make decisions such as this will be by beginning to design the product itself and treat development as an evolutionary software process using the material reviewed in this chapter as a solid foundation to build on.

Figure 5: The RGB colour cube. Notice on the cube on the furthest right, the corner closest to us is (0, 0, 0) and the opposite corner (1, 1, 1) represent black and white respectively. Images reproduced from [29]
3 Software Process and Requirements

On completion of the project, a list of basic requirements must be referred to so that the success of the design and implementation can be assessed. It is with this in mind that before designing the system, we need to clarify exactly what it should be capable of doing. This is more than simply making a list of aims but an opportunity, at an early stage, to ensure the structure that the construction of the project adapts is an efficient and valid one.

3.1 Introduction

In order to draw up a useful and obtainable set of requirements, we first need to determine the software process style we are going to adopt. Once this is determined we can begin to construct the requirements, utilising the knowledge specific to the problem, which was learnt in the literature review.

3.1.1 Adopting an evolutionary model

A software process model is an abstract representation of a software process [35]. It is important to adopt a method which will suit the problem and complement the research being undertaken. There are several models which we could follow, for example, the waterfall method of software design. The principle is simply that on the testing of the system (following standard construction of specifications and subsequent implementation), improvements are made to the system by reverting to the specifications stage. They are adapted and improved on the basis of the performance of the product. The implementation, validation and verification of the improvements is an ongoing process.

Despite its advantages, this system is perhaps not quite as flexible as the system in question will require. An evolutionary model of development (Fig. 3.1) will provide even more opportunity to redefine specifications.

The advantage of this method is that once a rudimentary system has been constructed, it can be easily adapted, manipulated and improved upon on the basis of the results obtained. If the project is based upon a set of relatively

![Figure 3.1: The Evolutionary Model of software design](image-url)
independent deliverables, such a system lets poor performance in one area be restructured. Equivalently if, in the design process, a particular method appears more favourable at the expense of a certain specification, that method could be implemented in the next generation, and then undergo the appropriate testing.

This model is certainly beneficial to a system reliant on a set of independent deliverables. In trying to decide between the suitability of two competing methods, such a development system easily lends itself to the independent testing of both. Plus the investigative nature of this project would certainly benefit from such a process. A project plan was constructed accordingly (Appendix C) splitting time between the development of the core system and implementation of refinements. Adherence to the plan was not intended to be essential, and leaving flexibility in the construction is fundamental to the evolutionary design process.

### 3.2 Defining System boundaries

One way of developing the scope of the system, and defining the boundaries of what it will be capable of, is to develop use cases. This requires considering how the user will interact with the system. In the introduction (Chapter 1) we saw that the immediate application for a project of this nature would be in the mobile camera-phone market. However, actual implementation in this way is technologically difficult to achieve with the given resources, so a desktop PC application will be built to demonstrate the process. In either case, user interaction is kept to a minimum. The user submits a query image and is returned a location – possibly with some additional information. Such a basic level of interaction means that at this level, we need only consider some of the most basic human-computer-interaction (HCI) principles. The system should ideally be able to identify the building in real time and have a simple to use graphical interface, although, as the main aim is the successful identification and categorisation of the building, these features should not come second to that target. The idea of identifying which features of the project are more essential to the finished product is the next step in requirements analysis.

### 3.3 Identifying and Prioritising Requirements

At this early stage in the evolutionary process we need to construct a set of “catch-all” requirements. We should first develop a set of specifications which are the most basic list of processes the system should be capable of. We should think of it as a list of “requirements for evolution” – a set of requirements, which while achievable in their own right, should also be capable of being adapted and expanded upon for future iterations of the process.

Because of this it would be useful if, in addition to the standard Functional and Non-functional categories, the requirements generated should be further categorised into those that are essential and those that are optional. Requirements flagged as high priority would be the most basic specification for the first stage of evolution, while those marked as optional could be implemented at this stage, or later as part of the intermediate versions.

This section deals with the capture of the requirements, with particular emphasis on what our primary system should be capable of. These can be split into
Functional and Non-functional requirements. We derive these from the problem statement and they are prioritised accordingly.

### 3.3.1 Functional Requirements.
Table 3.1 (over) shows the functional requirements in order of priority

### 3.3.2 Non-Functional Requirements
Table 3.2 (below) shows the non-functional requirements in order of priority

<table>
<thead>
<tr>
<th>No.</th>
<th>Requirement</th>
<th>Rationale</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>It should contain a database large enough to be of use on the University Campus</td>
<td>This is certainly something which will not be implemented till a later iteration of the system. To quantify “large enough” in this context is difficult. It would be sensible to define campus buildings as the front facing sides of those along the main University parade, the SU shop area, the buildings leading to and around the main bus stand and the larger buildings around the South entrance. An estimate of 40 buildings is a sensible success metric.</td>
<td>High</td>
</tr>
<tr>
<td>9</td>
<td>It should be capable of successful building identification within three minutes.</td>
<td>For the system to be of any kind of practical use, it should be capable of building identification in real-time. With the intended camera-phone application, the system should be capable of running within three minutes. However, to put this point as high priority would be misleading. The primary aim of the project is to demonstrate such a system is possible, so for that reason, in an efficiency verses accuracy trade off, we must lean towards accuracy on most cases.</td>
<td>Medium</td>
</tr>
</tbody>
</table>
Table 3.1: The Functional Requirements for the system

<table>
<thead>
<tr>
<th>No.</th>
<th>Requirement</th>
<th>Rationale</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The system should be able to take a photograph of a building and describe its location on campus based on a database of images.</td>
<td>This is the overriding aim of the project, and is essential to the primary product. The database should begin with a limited number of images, but should be created and maintained in a way that allows it to be easily expanded for future iterations.</td>
<td>High</td>
</tr>
<tr>
<td>2</td>
<td>The program will take a query image (using a digital camera), use a feature detector and compare features of the query image with those in the main database.</td>
<td>Feature detection was a major topic of the literature review and is certainly essential to such a project.</td>
<td>High</td>
</tr>
<tr>
<td>3</td>
<td>It should be able to compensate for a significant change in viewpoint from the database image.</td>
<td>This is another basic requirement of the project. Methods for doing this were discussed in the literature review (Section 2) and should be implemented with a view to satisfying this. In Section 3.4 we will discuss how to define success in this area.</td>
<td>High</td>
</tr>
<tr>
<td>4</td>
<td>The program will treat the building facades as planar objects</td>
<td>This was discussed in the literature review. To treat each building as a planar object significantly simplifies the problem and lets us use techniques such as RANSAC.</td>
<td>High</td>
</tr>
<tr>
<td>5</td>
<td>It should be robust and resistant to noise and clutter.</td>
<td>This should be considered from the earliest stages of the design in order to maximise the usefulness of the product.</td>
<td>High</td>
</tr>
<tr>
<td>6</td>
<td>It should be able to compensate for use with a poor quality camera, under poor lighting conditions or significant changes in illumination.</td>
<td>A key aim of the overall project is that the user should be able to achieve good results even in poor lighting (such as night time, and under the casting of shadows). This is where the evolutionary model will be useful, with such aspects being researched once a system capable of reliable feature detection is established. However, this important feature should not be totally ignored and then dealt with in isolation later. The project must develop in such a way as to easily allow advancement in this area to be made at a later stage.</td>
<td>Medium</td>
</tr>
<tr>
<td>7</td>
<td>It should be able to compensate for images of poor resolution images</td>
<td>640 x 480 is usually the highest quality image a camera phone is capable of and as such, it should be able to cope with images below this resolution.</td>
<td>Medium</td>
</tr>
<tr>
<td>8</td>
<td>It should have a suitable user interface</td>
<td>As mentioned when discussing User Cases (Section 3.2), the usability of the system is important, and while not the aim of the project, it should still be user-friendly, possibly incorporating a GUI.</td>
<td>Low</td>
</tr>
</tbody>
</table>
3.4 Testing and Evaluation Plan

With a project of this nature, there will certainly be images that the system will not be able to identify. However, we need to define what we will regard as a success. The requirements point to the following independent variables which will require systematic testing:

- Point of View
- Resolution
- Illumination

In addition to these, we must also pay attention to the program’s running time. In developing a suite of images with which to test the program, we need to ensure that they are capable of accurately representing the above, continuous data sets in a distinctive and useful way.

Viewpoint invariance is an important feature so buildings at a range of angles need to be amongst the test data.

The method by which the data set used to demonstrate viewpoint invariance is illustrated in Figure 3.4. A square grid will allow us to assess the quality of the result at a range of angles and distances.

When testing illumination, lowering the contrast of an image is not the same as taking a picture in the evening due to atmospheric conditions and their impact on the workings of the camera. For this reason, both will be looked into, with the camera being used in the same location at, say, 3 hourly intervals. The camera used for the database images must be a constant and the internal parameters of the camera (zoom functions, etc.) must also be unchanged. Collections of images of this nature can be used as a “test harness” with which we can test successive versions of the product. Sets of photos intended to test viewpoint invariance, illumination and resolution are seen in Figure 3.3.

Defining success metrics with a project like this is difficult. While it is easy to define a quantity such as angle, to pick a threshold of “pass” or “fail” is more difficult. The aim is to make a product that will work under a range of illuminations and angles and identify the building correctly in a reasonable amount of time.

![Figure 3.2: How a 2mx2m grid can provide a range of test angles](image)
3.5 Resources

In a real-world situation, the user would submit the image by camera-phone. As such an implementation is unlikely to be achieved, it is better to specify the user inputs a query image (for our purposes: into a desktop computer, running MATLAB 6). As discussed earlier, image processing in MATLAB is suitable for this project due to it’s intuitive format being suitable for rapid prototyping. However, this is marred by the fact that it is slow in terms of execution.

For this reason, it would be preferable if the user is kept informed of the processes being performed by the program on a regular basis (not only does this prevent the user from suspecting there has been an error, but will easily indicate to the programmer which parts of the process are particularly slow). Messages will depend on the design of the system, but messages such as “Searching…” or “Checking Library…” would be enough to ensure the user is aware the process is underway.

Figure 3.3: Selections from the test set showing standardised changes in viewpoint, illumination (by time of day) and resolution.
4. Design of Core System

An evolutionary system requires the construction of a single working model in order to assess the success of its individual components. These can then be tested and improved upon. In this section we develop the initial core system. Although at this stage the system does not contain a large database (it currently only comprises of 3 images), the system highlights the areas which will require more work and demonstrates the suitability of the basic theory behind the project.

4.1 Architectural Design

In this section we study the high level design of the initial system we implemented. The algorithm developed is capable of being split into five separate components, each of which is solvable via standard Computer Vision techniques. However, their interaction is intrinsic to the success of the project. The components of the algorithm are:

- **Corner and Feature detection**
  Feature detectors are applied to the query and database images. These will be used to find the homography between the images.

- **Finding the Homography**
  As discussed in the literature review (Section 2.4.2), a homography is a 3x3 matrix which represents the relationship between two images – effectively representing the change in viewpoint. For planar objects (which are what we assume the buildings to be) we can compute the homography by finding four corresponding points in the two images. This search is performed via RANSAC, to minimise and error function, (Discussed in detail in Section 2.4.2). The method by which this was eventually implemented is discussed in Section 4.2.1

- **Mapping**
  By applying the homography to every pixel in an image, we obtain a warped version of the image. In the event of a correct match, the two images should strongly resemble each other. Mapping techniques will be discussed in section 4.2.2.

- **Error Checking**
  A suitable metric would need to be drawn up in order to assess the quality of the match and therefore determine a winner from a list of possible buildings. Section 4.2.3 covers the method of error checking chosen for this system, and the development of a “mask” applied to the database images to improve the quality of the results.

- **Making a database**
  A suitable system needs to be constructed in order for the program to efficiently cycle through database images and assess the quality of the match in each case. In the earlier systems, this is less important as a smaller database will be used. With a larger system, effective hashing techniques will be required. Section 4.2.4 studies the versatility of Matlab
“structures” and how they can be implemented in this way, though hashing techniques will be left to Section 5.

![Block diagram showing method of core system](image)

Figure 4.1 Block diagram showing method of core system

This choice of method is based strongly on the research conducted in the literature review (Section 2) and, as figure 4.1 shows, is easily split into individual modules making the updating of individual aspects of the project in isolation of the other sections easier to deal with.

### 4.2 Implementation of basic system

The implementation of the core system tends to take an approach of “path of least resistance” to the problem. In almost each case the method which is eventually chosen is the most intuitive and easy to implement. The overall aim at this point of the development was to produce a system capable of basic image searching. The description of its implementation process broadly follows the same order as described above.

#### 4.2.1 Feature Detection and the RANSAC algorithm

As mentioned above, much of this method is based on well known Computer Vision techniques. There are several libraries almost standard functions which could be called on. Peter Kovesi of The University of Western Australia has produced a useful and popular set of Computer Vision functions to be run in MATLAB which are available to download from his website [36]. These provided an interesting starting point, however, as runtime was likely to become a problem, it made sense to ensure the functions were tailored to our needs and with that in mind, a set of problem specific functions were created. These were based on the literature review and previous implementations (such as those by Peter Kovesi), but allowed ourselves to target problem more specifically in the code and omit procedures of no interest to us, which could potentially slow down the process.
Feature detection is a standard of computer vision and a method for its execution is described in the literature review (section 2.4.1). The method implemented in the code is a simplification of Philip Torr’s Harris Corner detector [37] (the main simplification being it stopping short of sub-pixel accuracy).

Briefly, the process is, following a greyscaling of the image, to use convolution (the mathematical blending of two functions) twice. Once on the image with the mask

\[
M = \frac{1}{3} \begin{pmatrix}
-1 & 0 & 1 \\
-1 & 0 & 1 \\
-1 & 0 & 1 \\
\end{pmatrix}
\]

and the second time, apply the transpose of the mask to the greyscale image. The effect is to create horizontal and vertical gradients of the image, which (following smoothing with an appropriate Gaussian filter) can be used to compute an image which gives every point a value for its “cornerness”. The top, say, 200 of these can then be selected – and their co-ordinates can then be used in the RANSAC algorithm. These collections of points, together with the greyscale images, are then entered into the function matchbycorrelation.

matchbycorrelation (the only code of Kovesi’s in the final implementation) outputs 2 sets of points (m1 and m2), which are putative matches. As such, it is thought that \( m1(i) = m2(i) \), and thus makes the performance of RANSAC easier. Matchbycorrelation places windows around each Harris-detected point and calculates each point’s correlation strength against every other point. These are stored in a correlation matrix and the points are assigned to each other on the basis of matches in both directions.

The RANSAC algorithm (part of the code in the Appendix) follows the method described in section 2.4.2. The four points selected in each image are already putative matches thanks to the matchbycorrelation code. This is run a maximum of 1000 times (less in cases in which the probability of the current leading homography to be correct is overwhelming). Following the finding of the homographies between randomly selected pairs of sets of points, each potential homography is assessed for it’s quality by applying it to the whole set of points and assessing how many fall within a pre-defined threshold (which works successfully with a value as low as 0.003).

This part of the code broadly remained undeveloped throughout the project though parts were re-written in successive iterations of the program in order to make it more orientated to the problem in hand. In particular, the way in which RANSAC marked the “winning” homography became intrinsic to the overall scoring system which was adopted – though at this point in the development, the whole process was purely a means to finding the optimal homography \( H \).

Any two images will allow a homography to be calculated, however, for totally disparate images, the resulting 3x3 matrix will be meaningless. It is for this reason that we attempt to apply the homography to one of the images to assess the quality. Figures 4.3 and 4.4 clearly show that a correct and incorrect match is clear to the naked eye – and it was by looking at images of this nature that the various thresholds used throughout this stage of the code were able to be determined (for example, the number of corners being searched for was
increased from 200 to 300 and for an unnoticeable increase in computation time, the quality of the images rose dramatically. Once transformed we must find a way of assigning a numerical value to a homography quality. Before all this, however, the image needs to be efficiently transformed.

4.2.2 Performing transform
Following the determination of the homography, H, which relates the two images, we must use this to construct a method of determining the level of success of the homography. In order to do this, we need to apply the homography to the image – creating a warped version of the image which we can compare using a suitable metric.

The homography will take a 2D homogenised point (i.e. \([x\)-co-ordinate, \(y\)-co-ordinate, 1]) and multiply by H to find its corresponding point in the other image. However, in order to avoid unwanted effects caused by rounding errors, a process known as backwards mapping was implemented. This involves using the homography to see what colour the destination pixel should be coloured (as opposed to the other way round). This means applying the homography to every point on the first image and seeing the colour of the pixel it corresponds to on image 2.

The method above describes a system which provides reasonable results. However, it is unusual for the homography to send us to an integer valued pixel which we can easily take the colour of. More likely, we are directed to a decimal point. The round function in Matlab solves this problem to a reasonable degree with the x and y components rounded to the nearest integer so the colour can be safely assigned. This is a type of Interpolation Algorithm, in particular one called “Nearest Neighbour” as its dependence on rounding means that the new pixel is dependent on the nearest pixel to where it is mapped to.

Figure 4.2 demonstrates how problems arise with points such as \((2.5, 2.5)\), which could take any of the colours of the four surrounding pixels. Bilinear interpolation takes values for the four surrounding pixels (effectively up, down, left and right) but takes a weighted average of the 4 based on the ‘position’ that it is trying to map to. For example, consider the point \((2.8, 2.9)\)... We consider the X and Y co-ordinates and take the weighted sum for each. In the case shown in figure 4.2 the equations would be

\[
v_{\text{vert}} = \left( \begin{array}{c} r_{3,2} \\ g_{3,2} \\ b_{3,2} \end{array} \right) \times 0.1 + \left( \begin{array}{c} r_{3,3} \\ g_{3,3} \\ b_{3,3} \end{array} \right) \times 0.9
\]

\[
v_{\text{horiz}} = \left( \begin{array}{c} r_{2,2} \\ g_{2,2} \\ b_{2,2} \end{array} \right) \times 0.1 + \left( \begin{array}{c} r_{2,3} \\ g_{2,3} \\ b_{2,3} \end{array} \right) \times 0.9
\]

\[
v = 0.2v_{\text{horiz}} + 0.8v_{\text{vert}}
\]
Figure 4.2: Nearest Neighbour and bilinear interpolation methods

Notice we’ve made two interpolations (hence the name Bilinear) in the x and y directions and, as such, a program utilising this method will take much slightly longer to run than the Nearest Neighbour method. Even more time consuming is the Trilinear Method, which relies on constructing “Minimaps” of the image – the first Minimap being a halved copy of the original image, the second being half of that and so on. We then interpolate between levels by running the Bilinear algorithm on two of the levels and interpolating between the two levels themselves. For example, running the algorithm on the 1st and 2nd levels (that’s 2x and 4x) an interpolating would give a so-called “downsample image in 2.5 time”.

Given the length of time Bilinear would take, to try and run the Trilinear Method in Matlab on an image of reasonable size would take far too long. Bilinear was implemented, however, as such a method increases accuracy in the overall score the image will gain at the end.

Figure 4.3: (Left to Right) The database image, the query image and transformed query image.

Figure 4.4: Mismatched images and the resulting warped image. The next stage is to look into methods of comparing these against the database images.
4.2.3 Calculation of error

Following on from our manipulation of the query image in order for it to resemble the database image, we must find a way of assessing the quality of the transformation. The most intuitive way to do this is to compare the original image and its warped counterpart on a pixel by pixel basis. Up till now, we have been considering the warping of a pixel but not mentioned the actual pixel content. As discussed in the literature review, there is a range of colour spaces to choose from. We will look into the various advantages and disadvantages of other colour spaces in the refinements section (Section 5), however, the most intuitive (and MATLAB’s default) is RGB space.

In RGB space, a 640x480 image is stored as a 640x480x3 matrix, with each of the 3 dimensions representing the level of red, green and blue at that point. It is these values that we need to compare. We perform mean square error on each pixel, evaluating the Red, Green and Blue components according to the formula

\[
MSE = \sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2}
\]

This will give a mean squared error for each pixel. A perfect match (two identical pixels) will provide a result of 0. Two totally difficult pixels (in the case of RGB that is white against black) will give a result of \(\sqrt{3} \). It is easier to compare the buildings in terms of percentage – which could be done by simply representing each pixel as a percentage and then averaging the scores of all the pixels. This gives us a standard way to compare all the images and thus determine a best-match.

A problem arises in the region the image is mapped to. It is unlikely that the warped picture will take up the whole of the new canvas and we expect there to be pixels which will be left blank, as nothing is mapped to them. However, when it comes to averaging the whole image, there is clearly an unfair weighting. A good match in which only half the picture overlaps will obtain a poor score due to the fact it is averaged over all pixels, most of which will be compared to a black square. This could be beaten by a poorer quality match which covers the whole of the canvas.

![Figure 4.5: The transformed image is subtracted from the database image and the average pixel value is calculated – so black regions indicate good matches. The patch of full colour on the right – where the two images do not overlap prompts the development of the mask](image-url)
This can be easily solved by the construction of a mask in tandem with the new image. For every pixel successfully mapped into the new image, on a separate image we plot a white square (a pixel with the value [1,1,1]). Alternatively, if our mapping suggests that the pixel belonging at a point outside the boundary of our image then in our mask we draw a black point (a pixel with the value [0,0,0]).

Prior to the calculation of mean square error, we point-wise multiply the original image matrix with the mask. This means both our original matrix and our warp of the query image will now have the same region omitted. As long as a count is kept as to the size of suitably available pixels then we will be able to divide by the correct number of pixels and therefore determine a percentage score which truly reflects the images. This method was implemented successfully and, as we shall see in section 5, is adapted to other colour spaces without any difficulty.

Figure 4.6: The mask (left) created in tandem with the warped query image (centre) and the result of point wise multiplication of the mask with the database image (right).

Figure 4.7: An incorrect match (left) in which the query image of Fresh has been transformed according to a homography calculated by comparing it with the library and (right) the effect of subtracting the two images.
4.2.4 Structure of Database
The efficient construction of a database in Matlab is an important feature if the project is to be successful on a significantly large scale. Although the initial system will only rely on 3 images (which could be easily written directly in the code) requirement 3.3.8 means we need to make a system which can be easily expanded. Rather than construct a database in the traditional sense, Matlab uses a concept called structures [38].

A structure in Matlab is capable of storing numbers, arrays and strings, and is therefore perfectly suited to this problem. In this respect, they are similar to the concept of cell arrays though differ as elements are addressed by names called fields [39]. This makes it very easy to call data from different buildings. For example, if our structure is called “campus” then we could store the file path of every building in the database as “campus(i).path” where i is the number associated with each building.

Before the structure can be created, we need to decide what it is going to be storing. There are certain objects such as “photo” (The filename of the location of the database image on the hard drive – a string) and “Location” (Again, a string with the location of the image for the eventual output).

There are, however, other objects that could be stored in the structure. Being capable of storing arrays, it would be possible to store the actual images as 480x640x3 matrices. However, this unsurprisingly led to the structure file becoming increasingly large – with little advantage computationally (the imread and double functions which it would be eliminating from the computation process are amongst the shortest of the procedures in terms of time). It was this flexibility in the range of data capable for storage that made this a sensible way of making the database.

![Diagram](image.png)

Figure 4.8: The Original Matlab structure. The addition of the “image” branch significantly increased the size of the file and was eventually omitted.
4.3 Testing

4.3.1 Constructing the Test Set

The program was tested using three sets of images – two each of the library, fresh and the Norwood tower. The three buildings are certainly distinctive to the human eye and because of their respective sizes the distances from the building to the position of the camera vary greatly in order to get the whole building in to the picture.

The pairs of photos comprised of one of the building taken directly in front of the building, the next at an angle. The former being the “database” images, while the latter were intended to be the ones which will represent the query images.

The procedure was carried out by the subsequent running of three programs (as a function test.m). The first, carried out the corner detection and RANSAC procedures using the code taken from Peter’s Matlab Functions adapted to suit the procedure. The second function warped the query image according to the homography and a mask was prepared and applied to the (un-warped) database image. The two were then compared using mean square error in RGB space via a function called imagecompare which takes two images and calculates a percentage similarity based on mean squared error.

The main aim of this version was to see how successful the scoring system would be. Rather than making a system which took the query image and cycled through the database images, the user inputted two images and was returned a score for the quality of their match. Initial misleading results inspired the construction of the Mask (described in section 4.2.3 and figure 4.6) which was quickly implemented. The results can be seen in table 4.1 and in more detail in Appendix D. It can be clearly seen that images of the same building at different angles were returning scores in the region of 10% higher than their mismatched counterpoints (75% and 85% respectively). While giving a score of 85% seems like a sensible result, a score of 75% still seems quite a large score for mismatched images.

To a certain extent, this is something that should be expected. The fact two photos are of different buildings, it does not automatically follow that the score is going to be significantly lower. The only way in which the score is determined is via the colour checking method – and because colours can be in similar areas of the spectrum, there is perhaps not the huge difference we would ideally like.

However, there is certainly a distinct difference between images we were aiming to be similar and those we were not. A score also let us judge the quality of the homography being determined and meant that we could use it to calibrate thresholds. A significant improvement was found when increasing the number of corners searched for from 200 to 300.

At 200 corners, on running the test code consecutively on the same pair of images, the resulting warped images were obviously different and therefore capable of a different score. 300 lessened the seemingly random element and obtained more consistent results.
The other threshold which was able to be tested at this time was the distance threshold. During the RANSAC procedure, the concept of what defines an “outlier” is decided. In general, a lower score is better – though, obviously, too low and we run the danger of rejecting a potentially high quality Homographies. In the event of them all getting the same no of outliers then just the first will be selected.

![Figure 4.9: The Original Test Set used to assess the program’s performance under viewpoint invariance.](image)

![Figure 4.10: Images which have been artificially altered in order to test the system under poor illumination (left) and resolution (right).](image)

<table>
<thead>
<tr>
<th>Image 1</th>
<th>Image 2</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fresh</td>
<td>Fresh (angle)</td>
<td>86</td>
</tr>
<tr>
<td>Library</td>
<td>Library (angle)</td>
<td>86</td>
</tr>
<tr>
<td>Norwood House</td>
<td>Norwood House (angle)</td>
<td>85</td>
</tr>
<tr>
<td>Fresh</td>
<td>Library (angle)</td>
<td>73</td>
</tr>
</tbody>
</table>
5. Refining the System

The construction of the basic system was encouraging from the point of view that the much of the theory behind the project was working. However, what had been created fell far short of a product. Certain aspects required a great deal of expansion, in particular the construction of the database of images. Additionally certain areas were still taking far too long and the influx of new images that the full database would impose could render the system unusable (based on the time the three image system took to run). This section describes the development and testing of new ideas to improve and expand the system.

This chapter concerns attempts to make the product more versatile, for example, by improving its ability to perform in poor illumination (Section 5.1), as well as its efficiency, in attempts to hash the database using colour space (Section 5.4). Extensions relying on feature detection (both edges and corners) were eventually implemented for their contribution to the accuracy of the system (see Sections 5.2 and 5.3) as well as their ability to combat the effects of illumination invariance. As such, they are considered separately to the Illumination Invariance section, despite also aiding progress in this field. The chapter concludes with the differing aspects being assigned weightings on the basis of their performance in various tests and a hybrid system being constructed.

5.1 Illumination Invariance

The importance of colour space in relation to the problem cannot be underestimated. Requirement 6 states how the product should be capable of performance in varying illuminations so we need to ensure that we accommodate for such a situation. In the literature review (Section 2.5) we saw that in a similar system by Cipolla and Robertson [1] pixel intensity across the image was normalised. Here we look at differing colour spaces and how this could prove advantageous to the project.

5.1.1 HSV space

Figure 2.4, in the literature review (Section 2.5) showed HSV space as a cylinder of varying colour (Hue representing colour and the Saturation representing each colour on a scale of “Grey”). This is not strictly true, and the space is usually thought of as a cone due to the similarity of the darker colours as V → 0.

This differs from RGB space, which is usually thought of in terms of a cube of red, green and blue intensities. The advantage to the HSV is that, in moving in the V direction we are only altering the brightness. As such, the V is closely related to illumination so is likely to be useful in satisfying requirement 6. The natural question which arises is, in removing the V component from our calculations, is the whole system more robust to illumination?

To understand the impact the V component has on the image, it is important to view the effect of changing the component has on an image. Figure 5.1 shows a
picture of 3 East and how it appears at a range of V values. The images were created by removing the 640x480 matrix relating to the V component and replacing it with a matrix of uniform V values. The effect that value has on the image is striking with the features in the images being at their most clear in the mid-range and the colours gaining a kind of uniformity.

It’s resistance to changes in illumination is exposed in Figures 5.2 and 5.3. The poor illumination images were taken in particularly poor weather conditions – which are clear from the original image.

### 5.1.2 HSV vs. RGB

The original system constructed in Section 4 used RGB as a colour scheme. To make the change from RGB to HSV throughout the code is aided greatly by the fact that movement from space to space in Matlab is incredibly easy and fast, via the inbuilt functions rgb2hsv and hsv2rgb. The speed of these conversions is a massive boon to the overall re-coding of the project as it avoids a total rewrite of code previously written to accommodate the different space.

For this reason, we are able to introduce the HSV images at the point where the images are compared, making the RGB to conversion HSV following the warping of the image. Minimising the amount of time spent in HSV space significantly minimises the troubleshooting of any errors which may arise in the conversion process.

Section 5.1.1 discussed how a main advantage of HSV space is the strong reliance on the V component when regarding illumination. For this reason, the comparison code can omit V from its calculations. As such, the mean squared error averaging code was amended as the maximum score an individual pixel could achieve was now $\sqrt{2}$ (comprising of only the H and S components) saving a small amount of calculation.

At this stage, a significant disadvantage of the HSV conversion became apparent i.e. its treatment of black and white pixels. In the implementation of
both the RGB and HSV comparison systems, the initial test was to use in two identical images where we expect a similarity of 100%. Similarly, in order to test the most disparate cases, a totally white and totally black bitmap image was entered where a score of 0% would be expected.

This test failed in the HSV case, claiming the black and white images were identical. This can be explained by the removal of the V component from the similarity calculation. As figure 2.4 in the literature review displayed, white and black are on opposite sides of the V spectrum (as opposed to RGB where they are the total presence and the total absence of all three components, respectively).

Despite this, results for the HSV colour system were generally good (See Table 5.1). The right image was always identified – with an almost identical success rate for the RGB version (good results were in the 87% region, poorer ones in the 73-80% region). Again, the line between “good” and “poor” was slimmer than expected, although the white/black crossover could go some way to explaining this.

Table 5.1: Results of HSV tests

<table>
<thead>
<tr>
<th>Query Image</th>
<th>Fresh (Database)</th>
<th>Fresh (angle)</th>
<th>Norwood</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fresh (Database)</td>
<td>100</td>
<td>87</td>
<td>74</td>
</tr>
<tr>
<td>Fresh (angle)</td>
<td>74</td>
<td>79</td>
<td>86</td>
</tr>
<tr>
<td>Fresh dark (artificial)</td>
<td>77</td>
<td>67</td>
<td>56</td>
</tr>
<tr>
<td>Fresh dark (real)</td>
<td>74</td>
<td>73</td>
<td>73</td>
</tr>
</tbody>
</table>

Another test was to see if the point at which we cross over to HSV space mattered. Tests were done in which the image was warped prior to the code which performed the homography. The results were almost identical – with the winners gaining scores as little as a decimal place higher or lower between the two systems (losing buildings underwent slightly more fluctuation, though this is probably due to the inherently random homography being applied to the image, which may result in a range of possible “best matches”).

The suggested advantage of HSV space was its performance in poor illumination. There are two ways to test this. The first is to artificially decrease the illumination of the image via a suitable image package. This is a very clinical way of testing the problem – it will show performance under controlled, uniformly administered conditions. However, such a system is not reflective of images actually taken in poor illumination conditions, so a query image was taken in particularly poor conditions (regarding both illumination and weather!).

Table 5.1 also includes both of these results, and again the correct building wins in each case – despite significant deviation from the database image (see Figure 5.3). The margin of error, which we have already commented on as being narrower than one would expect, was particularly slim in this case.

Such a small difference between buildings being declared the winner or loser is far from ideal, however, we need to assess how this compares to the alternative system – RGB. In the case of the artificially degraded image, the results showed a stark contrast with RGB scoring 54% and HSV 74% - though both
winning their respective fields. Of more concern was the tests of fresh actually taken in poor illumination, where the RGB system identified it incorrectly as the library.

5.1.3 Conclusion
While RGB and HSV both perform well in good illumination, in the event of choosing just one, HSV is the natural choice due to its ability to cope better in cases of poor illumination. There is certainly an argument to use both systems and construct an average, or weighted average – however, requirement 10 clearly states that we have a duty to present a system which works within a reasonable timescale and the advantage which we would gain from using both cases is not worth employing the extra computation time.

5.2 Edges
As explained at the start of this chapter, the idea behind studying feature detection in system refinement is twofold. Strong features in an image will be detectable in poorer illumination and as such will combat the issues surrounding inadequate lighting. This is particularly important following the results of section 5.1 which, although showing that HSV is the sensible choice for a range of lighting scenarios, the margin by which buildings were declared winners was worryingly small. The second reason is that edge detection could be used as a method of comparison in its own right. In this chapter we implement an edge detection system and subsequently decide that though not reliable enough to form the entire basis of a comparison system, would certainly be worth implementing in a weighted comparison system.

5.2.1 The Sobel Method of Edge Detection
In computer vision, an edge is a discontinuity in an image. It is expected that at the boundary of an object in the image, there is a sharp change in colour. Edge detection methods highlight these changes in colour by considering horizontal and vertical scan lines of intensity of the image. At a discontinuity the change of
colour is represented by a large change in gradient. It is these scan lines which are then combined to give an edge detected version of the image.

There are several different edge detection methods carried out by the process of convolution. The effect is of a small window scanning the image, highlighting the changes in gradient. The method adopted was the Sobel operator. Two matrices scan the image:

\[
\begin{align*}
K_{\text{horiz}} &= \begin{bmatrix}
-1 & -2 & -1 \\
0 & 0 & 0 \\
1 & 2 & 1
\end{bmatrix} \\
K_{\text{vert}} &= \begin{bmatrix}
-1 & 0 & 1 \\
-2 & 0 & 2 \\
-1 & 0 & 1
\end{bmatrix}
\end{align*}
\] (1) (2)

the horizontal and vertical matrices, respectively. An edge will be defined as the sets of points in which discontinuities are discovered in both the x and y directions, and as such, the edge detected image is the result of considering the maximum of the image convoluted by \(K_{\text{horiz}}\) and \(K_{\text{vert}}\) on a pixel by pixel basis.

Figure 5.4 shows the convolutions with the two window matrices and how they combine to give an edge detected image. The implication to this project is that this is a potential alternative category of image to use in the comparison stage of the proceedings. We have already used two colour spaces – and seen how they perform in varying illumination. How do edge detected versions of the images compare?

Initial tests in full illumination were good. The same test set were used as in the HSV and RGB cases – with, for example, an image of Fresh at an angle gained the score of 85.5%, while others were in the region of 75%. This margin of success was not repeated in all tests – and in some cases, the buildings were all so close together that the wrong building won.

![Figure 5.4 Images with Horizontal (left) and Vertical (centre) Sobel matrices applied and the combined edge detected image (right).](image)
This may be explained by the method’s poor resistance to clutter. In computer vision clutter is defined as anything in the image which we are not interested in [11]. As you can see from the pictures in fig 5.4, people walking past are emphasised by the edge detector and, as such may significantly weight the score. This is a problem and as such, may mean that edge detection should be used as part of a larger scoring system, and perhaps not by itself.

5.2.2 Edge detection and HSV Space

The main advantage with edge detection is its ability to continue to provide reliable results in poor illumination. Figure 5.5 shows a set of images which have undergone an artificial change in illumination while figure 5.6 shows how well they react to Sobel edge detection. The images were taken in daylight, and then the contrast of them was artificially lowered. In line with the deterioration in the image, the definition of the edges is also lost, however, in these cases the edge detection has been done directly on the regular RGB images. We saw above how HSV deals with image contrast differently, and the effect on edge detection was investigated accordingly.

The method used was to take the three images, and transform to HSV space. The V component was then made uniform across the image by replacing the entire V components by a 640x480 matrix of 0.5 (Figure 5.7). These images then underwent the same edge detection procedure as described above. The images obtained (Fig 5.8) show a marked improvement on the RGB images. Although we have encountered a great deal of noise in the process, the definition and quality of the HSV images are far superior to their RGB counterparts.

5.2.3 Conclusion

Table 5.2 shows the results of the edge detected images. The edge detected images do not perform as well as the colour based methods seen previously (there is one failure), however, the edge detection in poor illumination the system has certainly proved itself and as such should not be discounted. In the event of a system which uses weightings of a variety of methods, this would certainly be one of them.
Figure 5.5: A query image of Fresh (right) and the same image having undergone artificial decreases in illumination.

Figure 5.6: The images of Fig 5.5 having undergone Sobel edge detection

5.7 The image of Fig 5.5 having undergone normalisation of the V component (as described in Section 5.1)

5.8 The images of Fig 5.7 having undergone Sobel edge detection. Notice the increased definition of the edges compared to 5.6

Table 5.2: Edge Detection Tests

<table>
<thead>
<tr>
<th>Image Entered (each is the building at an angle)</th>
<th>Winner</th>
<th>(Score)</th>
<th>Second Place</th>
<th>(Score)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fresh</td>
<td>Fresh</td>
<td>(86.7%)</td>
<td>Norwood</td>
<td>(75%)</td>
</tr>
<tr>
<td>Library</td>
<td>Library</td>
<td>(75.8%)</td>
<td>Norwood</td>
<td>(69%)</td>
</tr>
<tr>
<td>Norwood</td>
<td>Fresh</td>
<td>(78.4%)</td>
<td>Norwood</td>
<td>(75%)</td>
</tr>
</tbody>
</table>
5.3 Corners

One of the features we have already looked at that is particularly resistant to illumination, and which is fundamental to the project as a whole, is the detection of corners in the two images. A major difference between edge and corner detection is the corner detector described in Chapter 4 works on a threshold basis. It searches for the 300 “best” corners, as opposed to edge detection, which has no such quota to satisfy. As such, in any image, the detector will find 300 distinctive points. The points in one image may be significantly less clear than in the other, but this is irrelevant once the corners have been identified.

In this section we shall look into corners and find that the method used to determine the optimal homography in the RANSAC algorithm (Section 4.2) can also be used as an aid to the identification of the building itself (based on the number inliers). However, the question which immediately arises is, if such a system is to be used, why are we not just using the minimised error for RANSAC to find the best match?

The answer becomes apparent in Section 5.3.2 when this method is tested. While some buildings are capable of being identified by this system alone, several fail to be identified correctly. Like edge detection, this must be used in collaboration with other methods – though the speed of the method, especially compared to the comparison techniques which require the warping of an image, marks it out as a possible hashing technique.

5.3.1 Corners and RANSAC

In Section 4.2.1 we saw the fundamental importance of corner detection in the project. The RANSAC algorithm uses sets of four random corners in each image, computes the homography between them and analyses the success of that homography by measuring the error function - the best one being the one that becomes the RANSAC-derived homography. The measure of success of this system is when the discovered homography is applied to the entire set of corners of one image, how many of these match corners in the corresponding image. This is done by comparing the points against a suitably small threshold (as small as 0.001).

It is clear that the smallest match possible will be four corners – since the homography is solely based on the successful mapping from the four corners in one image to the other. However, after 1000 trials, the homography which has up to this point gained the most matches (or inliers) within the threshold is the winner.

The fact that every image which we intend to warp will undergo this procedure means that before we have even got as far as warping the images, we have a possible success metric which we can refer to ingrained in the RANSAC code. It is a relatively quick process – certainly far faster than the computationally expensive comparison functions described above.

5.3.2 Development and Testing

There are several ways of attempting to construct a scoring system from such data. For example, using the “inliers” (the system described above) we calculate a percentage as the largest number of possible inliers is of course 300 (for
identical images). This worked very well with the initial, yet small data set – particularly images of Fresh, which would gain a score of up to around 40 or 50% while losing buildings would be as low as 5%. The other buildings in the set (the Library and Norwood house) while successful performed less well.

Looking at the corner-detected images easily highlights the reason behind this. The building Fresh contains a large corner – heavy logo that weights the points to a certain distinctive region. The fact that a single region so heavily weights the picture means that the corners in that area will also be successful and thus a high corner score is achieved.

The speed at which this particular method could be run made such testing extremely easy (especially in comparison to the warping comparison). It is this potential increase in speed that made this line of enquiry worth pursuing. Because of this, the range of images used for corner detection was expanded greatly.

Tests with a larger range of buildings were less successful in both cases as Table 5.3 shows. There were cases such as Fresh where there was a clear winner far in front of all the others, however, in most cases, the correct building came a close second or third.

Table 5.3: Highlights of Tests for Number of Inliers

<table>
<thead>
<tr>
<th>Query Building</th>
<th>Winning Building</th>
<th>Inliers Score (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fresh (Angle)</td>
<td>Fresh</td>
<td>48</td>
</tr>
<tr>
<td>4 East (Angle)</td>
<td>URB*</td>
<td>10.3</td>
</tr>
<tr>
<td>8 West Reverse (Angle)</td>
<td>8 East (Side)*</td>
<td>10.3</td>
</tr>
<tr>
<td>Chaplaincy (Angle)</td>
<td>Chaplaincy</td>
<td>11.67</td>
</tr>
<tr>
<td>Dolche Vita (Angle)</td>
<td>Dolche Vita</td>
<td>10.3</td>
</tr>
<tr>
<td>STV (Angle)</td>
<td>UBSA</td>
<td>12</td>
</tr>
</tbody>
</table>

*The starred cases indicates the correct building came in second place
For this reason a new system was drawn up which used this test as a stage in a larger process. Following a test of the number of inliers the buildings are put in a “league table” according to their score. The idea was that the top three would go on to further testing using HSV comparison methods (the only exception to this rule was any building with a score of more than 10% higher than its nearest rival).

The results were certainly more successful than if a straight forward “highest corner score” had won however in a few tests, the correct building was losing out for being in a tie break situation.

Testing all buildings which arrived in 3rd place buildings was only a partial success as, in some cases, by 3rd place every building in the database was being tested. It was becoming clear that each case had different requirements and a method was required that would be able to deal with a range of eventualities.

It was decided that, on ranking the buildings, to take the one with the largest score out of 300 and construct a percentage based on this being the greatest score. This meant the building in the lead would be 100% and the others would be scored relative to that. Trials were run in order to determine a threshold which would hash the database suitably yet allow a clear winner to go to pass through unchecked. Through trial and error, a threshold of 70% was determined as enough.

### 5.4 Improving Scalability of the Database

In the previous section we discussed the introduction of an increase in the size of the database. The procedures so far described were developed on a small (and deliberately distinctive) set of data due to the time taken for each of them to execute. The final system, however, will have to prove itself capable of coping with far more buildings if it is to demonstrate a usable product, in line with Non-functional requirement 9 and in a sensible time scale, according to requirement 10. It is for this reason we now look into effective methods of hashing the database.

In this section we look into ways of hashing the database, or more fittingly, eliminating the most unlikely. We continue to use our findings on colour space to find the average colour of a query image and eliminate the least likely buildings on these grounds. The result is a system dependent on a range of thresholds which run the danger of allowing faster, yet less accurate searches and must be carefully calibrated to ensure we deliver best system possible.

#### 5.4.1 Making the database

In the initial construction of the database in section 4, the amount and type of data the structure could contain was looked into. The basic system contained

- Image location on hard disk
- Name of Building

These pieces of information are the most fundamental possible, however, the Matlab structure facility is capable of the storage of a wide range of data – and
as such could hold information such as the image itself saved in Double form, greyscale of the image or even a list of the corners in the image.

In the case of the double and greyscale images, to save these in the database would certainly be counterproductive. To pre-calculate such a large amount of data may at first seem useful, however, the disk space required soon increases to an unreasonable level. The 3-image system alone had a size of over 5Mb by the time the 3 images were entered as 640x480 matrices. This was confirmed by the Matlab “profile” function which returns the time each individual function in a program takes to run. While the functions that perform the homography were taking minutes to run, the imread functions were taking only fractions of a second.

In the case of corners, the decision was more difficult. The time taken to corner detect was 79 seconds, however, to save to a database, it would require the saving of 30 co-ordinates, rather than over 300,000 colour pixels. It was decided to leave this for now – and rely on further hashing techniques prior to the corner finding process to resolve this.

5.4.2 Hashing the database

Hashing is the splitting of a database of objects into subcategories for the purposes of refining a search. In the case of this particular problem, hashing is important as the methods of image detection are computationally expensive and the program as a whole is in danger of violating requirement 9. As such, whatever techniques we employ must be fast.

A prospective hashing method was the appearance of a common colour. Since the most basic methods we have so far developed are all corner and feature based, it is feasible that the program could be testing a building of totally different colour throughout the process, only eliminating it at the very last stage. It became clear that a useful process would be to develop a way of eliminating buildings of obviously incorrect colour at a very early stage in the program.

We have already seen how colours can be mapped as a point in RGB or HSV space and how, for our purposes, HSV was the more useful due to our ability to only need to consider the H and S components. Thinking of H-S space as a 2D graph, the pixels of a query image could be plotted on an H-S graph.

![Figure 5.10: A graph of Hue against Saturation showing the value of every pixel in an image of 8 West 2.1. Notice the appearance of common colours](image)
The first method by which the database was hashed was by considering the average HS point for every image in the database. Then, the query image’s average colour value could be effectively plotted on a graph and all the buildings with an average colour within a suitable radius would be tested, eliminating the outlying buildings. Obviously, a large database of images is required in order to provide a worthwhile test and a collection of 52 images were entered into the database. An initial graph of the 52 averages (Figure 5.11) highlighted a problem that was expected. There was a significant amount of clustering. This led to a difficult trade-off in setting the threshold for the system i.e the radius around the query point we consider. It needed to be large enough to encapsulate the building we require, however too generous a threshold meant it quickly started to contain a large number of images. Tests suggested a threshold of 0.15 was enough to encapsulate the buildings we required however, the resulting data set still regularly comprised of over 10 possible buildings. Using images in poor illumination, the correct building made the shortlist with a threshold of 0.15, however, as part of a list of 37 images, subsequent elimination was clearly required to make the system practical.
It was clear that further hashing would be required, and RGB space seemed a logical way to progress. As you can see from figure 5.13, RGB 3-space saw similar clustering to the previous cases only in 3 dimensions. RGB's poor dealing with illumination conditions meant that the choice of threshold had to be quite broad - but due to the speed of the process it was still well worth implementing and even though the process of finding the average pixel value was quite fast, it was made even more efficient by saving the average HS and RGB co-ordinate as part of the Matlab structure. This made the whole process significantly faster.

The two processes together were successfully limiting the range we needed to test, though further elimination processes were required. Following on from the theory seen in the edge detection section (5.2) about the standardising of the V component in an image to enhance illumination, a system was developed in which an image was transferred from RGB to HSV, the V components normalised to 0.4 and then transformed back to RGB. Figures 5.1 and 5.2 showed how the images compare on being returned and how the colours are "standardised" in the process.

This appeared to be a great success. The entire range of buildings appeared in a close proximity (Figure 5.14) meaning a threshold as low as $t=0.03$ was originally thought to be suitable, however, as more buildings were tested, it was decided this needed to be greatly expanded. Each of these processes could be used in isolation, but their efficiency meant it was worth applying them all consecutively. Experiments implied that the two HSV systems also worked in poorer illumination, while RGB was less successful. For this reason, a clause was added to avoid the RGB test in versions of poor illumination. Since a poorly illuminated image led to the threshold of a query building lying too far from any of the building points, if the RGB list returned an empty list then it was ignored. Each of the lists made by the three systems were then combined at the end via the intersect function, therefore providing a short and efficient list.
Figure 5.14: The average points of all 52 images in the database in RGB following conversion to HSV space and normalisation of the V component.

The results demonstrated the success of this system against a wide range of demanding images. The overall success rate was good, however in several cases, the correct building was eliminated. In these cases, it was common for the correct building to make at least one of the lists, however, failure in one or more list led to overall failure. The obvious solution would be to change the intersection of the 3 lists to a union – however, the immediate effect was to dramatically increase the lists of candidate images – the exact opposite of what we had aimed to do.

This was an important trade-off of reliability against efficiency. We have already discussed the amount of time image processes take to run in MATLAB, and even if the system is meant to demonstrate the potential of the process rather than become a commercial product, to significantly increase the list of buildings that would require full testing would jeopardise the ability of the system to run in anything resembling real-time.

On returning to the requirements, Requirement 10 discusses trade-offs of exactly this nature and the priority of the requirements makes it clear that we should make a reliable yet potentially slow system. There were two ways to do this – replace intersections with unions throughout the lists, or increase the thresholds. While the union solution was the quick-fix, to spend time testing and calibrating the thresholds meant that we could more successfully target the problems with each of the systems. This was done by using the buildings which caused the most problems in our previous test set and find the threshold required to contain them.

The resulting system certainly proved more reliable – and the process was quick, however, for the system to be in any way useful, another quick method of elimination was required.

5.5 Building a combined system

Each of the techniques seen so far in this section are capable of being used in isolation in order to perform content based image retrieval on a query image. However, throughout the section no method has come across as being an
“overall winner” or a definitive method. The average colour techniques were developed with the express intention of hashing the database and are clearly implemented for their speed, as opposed to their accuracy. Equivalently, edge detection and pixel-by-pixel comparison in HSV space are designed to identify the exact image, even in adverse conditions – however, each had proved that they could sometimes make mistakes. The next stage was to combine all these techniques in the most effective way. This section discusses how a weighting system was developed and loose ends were tied up as the finishing touches of the code were put together.

5.5.1 Improving Efficiency

The introduction of thresholds in section 5.4.2 allowed us to impose a certain level of control over the accuracy and efficiency of the system. To lower the three thresholds described will increase the speed of the system, but lessen the chances of a successful identification. However, by increasing the thresholds we soon find ourselves in the situation of having to perform the full test on every database image.

As such, elsewhere in the code, every effort had to be made to save time. This is especially important as the modular aspect of the overall program and the way each section was effectively developed in isolation of the others, meant that there were sections of code being repeated. As such, the first process was to ensure everything regarding the query image became a global variable.

As mentioned previously, MATLAB contains the function “profile” which returns a webpage of the time each function takes to run and what percentage of the overall computing time it is occupying. A large development was the use of the structure to pass information between the individual modules of the code. For example, the optimal homography found as the result of the corner detection was capable of being saved to the structure and called if it was required in a later section of the code.

5.5.2 Developing a weighted function

Up till now, the system had progressed by the query image progressing through a series of “rounds” in which buildings from the database are either eliminated or progress to the next round. First the image is tested for its average colour and a shortlist of possible buildings is made. It then undergoes corner detection where we determine the “top 30%” which then go on to be tested via the image comparison methods in HSV.

Tests were carried out using this system and the results were poor with only 5 out of the 39 images tested being identified successfully. These tests were closely monitored so as to accurately pinpoint the point of failure in the hope that each stage could be modified further.

The first change was to modify further the thresholds of the colour hashing system, developed in section 5.4.2. This had an immediate effect on Requirement 10 due to the dramatic increase in the time the program took to run (which was already far too slow for the requirement). However, the change allowed for up to half of the buildings which had previously failed to reach the second stage. The interesting result was in buildings gaining a good corner score but eventually losing out to a similarly coloured building in the final stage.
It became apparent that original system of “pass and move on to the next round” was not rigorous enough. A good corner score needed to do more than just allow the building to be tested at a later round – it needed to have some bearing on the final score, and it was for this reason a weighting system was introduced.

The idea that the system was not going to revolve around one “final test” also allowed the re-introducing the edge detector system into the final tests. This had been eliminated based on the mentality that each module was nothing more than a means of progression to the next round. However, as part of a combined system, it made sense to use both colours and edges combined as tests. Corner detection was clearly an important part of the process too – so was chosen to be deserving of a heavy weighting early on. By a similar logic, the initial colour checking was not weighted as heavily due to the now significantly expanded thresholds eliminating the least likely, rather than selecting the most likely.

The initial weightings were 10% for colour testing, 30% for corners, 30% for edges and 30% for HSV colour checking and a marked improvement was immediately seen. Errors still implied a slight weighting towards the corner detection system was required and they were adapted accordingly. The final version’s weightings were

10% for original colour checking
40% for corner detection
25% for edge detection
25% for colour matching

The program was given a simple Graphical User Interface, satisfying requirement 8 developed using the Matlab Guide function. The user was invited to enter the filename of the image (starting from the current directory) and click “Find”. The result was for the building’s name to appear on the GUI.

![Building Finder GUI](image)

5.15 – The GUI is opened from the command line and provides a simple user-friendly way to interact with the system.
5.6 Verification and Evaluation

Throughout this chapter, there has been significant testing in order for us to assess which methods to side with and which directions to head in. The varying data sets have been chosen to highlight different issues throughout the project. However, the product is now in a state at which it can be thoroughly tested. As was described in the test plan (Section 3.4) the areas which we are particularly interested in are...

- Resolution
- Viewpoint Invariance
- Illumination

This section will highlight how the system performs under these conditions. As we shall see the hashing techniques and illumination invariance methods are in direct opposition to each other and, as such, the time the project takes to run will also be looked into. The Requirements, seen in section 3, will be called back to for reference to see if the methods implemented have been satisfied accordingly.

As much of the data we are assessing is continuous in nature (time, illumination, viewpoint invariance) the concept of “pass” or “fail” is a difficult one. Much of this chapter is testing until we reach failure. We shall find the project works well in angles of up to 40 degrees either side of the normal and resolutions as low as 80 x 40. Illumination invariance is successful up to 8pm on an evening in May—though the effect of shadows and the clash of interests with the hashing techniques are obvious and shall be discussed.

5.6.1 Resolution

Throughout the project images comprising of 640x480 pixels have been used, reflecting the image quality of a standard mobile phone camera. However, it is interesting to see how the system stands up to a range of resolutions. In this section we shall test the project with a sequence of decreasing resolutions until it eventually breaks – and we can find the exact point at which it fails.

In order to ensure we are only testing resolution, and no other factors are having an effect on the result, it was decided that our source image would be a copy of a database image. The image of the library (seen earlier in the project) was chosen due to its distinctive appearance and it’s even spread of corners and features across the image.

As resolution decreases, the definition of the image degrades and certain corners and features, essential to the successful determination of the homography, become obsolete. As such, we expect there to be a point at which the resolution becomes so poor, the features detected no-longer successfully represent the image adequately.

As the program has various parameters which are reliant on the image being of exactly 640 by 480 pixels, the test could not simply comprise of shrinking the image and then entering it into the program. The test set was created by a Matlab program which shrinks the image to a new resolution and then scales it...
back up. So for the first degraded case, the image was shrunk to fit 320 x 240 and then scaled back to 640 x 480. As is clear in figure 5.16, the difference in quality soon became clear.

When considering how to increment the testing, the methodology was to halve the height and width of the image with each successive image. The program used bi-cubic interpolation to recalculate pixel accuracy so as to produce images with were good representations of poor quality images. The method of degradation meant that by the first iteration, there were already one quarter of the number of pixels there were in the original.

This trend continued so the pixel ratios increased by powers of 4. As table 5.4 demonstrates, the first failure was 80 x 60 pixels where each pixel represented 64 in the original image (Fig 5.17). As such, requirement [7] is satisfied.

<table>
<thead>
<tr>
<th>Image</th>
<th>Res.</th>
<th>Ratio</th>
<th>Successful Identification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>640 x 480</td>
<td>1:1</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>320 x 240</td>
<td>1:4</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>160 x 120</td>
<td>1:16</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>80 x 60</td>
<td>1:64</td>
<td>N</td>
</tr>
<tr>
<td>5</td>
<td>40 x 30</td>
<td>1:256</td>
<td>N</td>
</tr>
</tbody>
</table>

Figure 5.17: The first image to fail and (left) a comparison of the 640x480 and 80x60 sets of pixels in the same area of the image, for comparison.
5.6.2 Viewpoint Invariance

With the constant use and re-use of the same data sets throughout the development process, there is a danger that the methods implemented would be tailored too specifically to the data sets seen throughout the project. For this reason, several newer data sets were introduced for the final testing, as well as those we had seen previously.

The methodology of testing was the same as that described in section 3.4 in which the query images would be standardised by using a grid of 2 metre squares in order to gain a range of angles and distances from the building. The testing was performed on a range of buildings and some of the more interesting results will be highlighted in this section.

The convenience store, Fresh, was one of the first 3 images in the database and has been a standard image throughout the project. For this reason, a new set of test images were taken. A line was drawn parallel to Fresh, 10m away from it, and pictures taken at 2m intervals along the line (measured, right and left, from directly opposite the door of the store). Table 5.5 shows the results obtained and the associated angles. Figure 5.18 shows the angle within which the result is successful, the final image within which the warp was successful and it’s associated database image. To further understand the cause of the failure, figure 5.19 shows the final successful image (taken 10m back and 6m to the right) and the image with the homography successfully applied, compared with the first failure.

![Figure 5.18: (Left) The maximum angle within which the Fresh was identified. The images (right) show the database image, and the building at the most extreme angle which was still successful.](image)

Table 5.5: Results of Fresh taken at differing distances and angles (see figure 5.18 for more information)

<table>
<thead>
<tr>
<th>Filename</th>
<th>Distance from centre (L= Left. R = Right)</th>
<th>Angle from Normal</th>
<th>Distance from centre of object</th>
<th>Building Correctly Identified</th>
</tr>
</thead>
<tbody>
<tr>
<td>fresh_10m</td>
<td>0</td>
<td>0</td>
<td>10 Y</td>
<td></td>
</tr>
<tr>
<td>fresh_10m_r2m</td>
<td>2 R</td>
<td>11.31</td>
<td>10.19 Y</td>
<td></td>
</tr>
<tr>
<td>fresh 10m_r4m</td>
<td>4 R</td>
<td>21.8</td>
<td>10.77 Y</td>
<td></td>
</tr>
<tr>
<td>fresh 10m_r6m</td>
<td>6 R</td>
<td>30.96</td>
<td>11.66 Y</td>
<td></td>
</tr>
<tr>
<td>fresh 10m_r8m</td>
<td>8 R</td>
<td>38.66</td>
<td>12.81 N</td>
<td></td>
</tr>
<tr>
<td>fresh 10m_r10m</td>
<td>10 R</td>
<td>45</td>
<td>14.14 N</td>
<td></td>
</tr>
<tr>
<td>fresh 10m_l2m</td>
<td>2 L</td>
<td>11.31</td>
<td>10.19 Y</td>
<td></td>
</tr>
<tr>
<td>fresh 10m_l4m</td>
<td>4 L</td>
<td>21.8</td>
<td>10.77 Y</td>
<td></td>
</tr>
<tr>
<td>fresh 10m_l6m</td>
<td>6 L</td>
<td>30.96</td>
<td>11.66 Y</td>
<td></td>
</tr>
<tr>
<td>fresh 10m_l8m</td>
<td>8 L</td>
<td>38.66</td>
<td>12.81 N</td>
<td></td>
</tr>
</tbody>
</table>
This level of success was repeated with the Accommodation Office, where the images were taken the same distance from the building. In this case, no failures were actually recorded, (as obstacles in the way made the range of positions it was possible to take photos from, limited). As is clear from the images and diagrams in figure 5.20, the results were resistant to the pillar which clearly enters the field of vision.

Such pillars are a regular feature of the campus structure and are successfully dealt with in the accommodation example due to the fact that they obscure none of the main features. The same could not be said of 8West 2.1, whose first failure coincides with a pillar clearly obscuring the main origin of the features in the image, the door of the building.
Figure 5.21: The database image of 8West 2.1 (left) and the same building at an angle of 34° and obscured by a pillar - the first image to be identified incorrectly (right).

Aside from the case of 8W2.1, where the pillars clearly had an adverse effect on the result, the angles within which the building can be successfully identified is starting to become clear. In particular, the buildings so far are, on the whole, successfully identified up to angles approaching 40 degrees. These figures are re-enforced by studying the results of buildings of a far smaller size. As such, to encompass the building in a single photograph, the location of the camera must be closer to the building and, as such, movement in each direction causes sharp increase in the angle at which the image is taken.

### Table 5.6: Results from URB

<table>
<thead>
<tr>
<th>Filename</th>
<th>Angle From Normal</th>
<th>Distance</th>
<th>Success?</th>
</tr>
</thead>
<tbody>
<tr>
<td>urb_2m</td>
<td>0</td>
<td>2</td>
<td>Y</td>
</tr>
<tr>
<td>urb_2m_r2m</td>
<td>45</td>
<td>2.83</td>
<td>Y</td>
</tr>
<tr>
<td>urb_2m_r4m</td>
<td>63.434948</td>
<td>4.47</td>
<td>N</td>
</tr>
<tr>
<td>urb_2m_l2m</td>
<td>45</td>
<td>2.83</td>
<td>N</td>
</tr>
</tbody>
</table>

### Table 5.7: Results from 4 East

<table>
<thead>
<tr>
<th>Filename</th>
<th>Angle From Normal</th>
<th>Distance</th>
<th>Success?</th>
</tr>
</thead>
<tbody>
<tr>
<td>4_east_2m</td>
<td>0</td>
<td>2</td>
<td>N</td>
</tr>
<tr>
<td>4_east_4m</td>
<td>0</td>
<td>4</td>
<td>Y</td>
</tr>
<tr>
<td>4_east_4m_r2m</td>
<td>26.57</td>
<td>4.47</td>
<td>Y</td>
</tr>
<tr>
<td>4_east_4m_r4m</td>
<td>45</td>
<td>5.66</td>
<td>Y</td>
</tr>
<tr>
<td>4_east_4m_l2m</td>
<td>26.57</td>
<td>4.47</td>
<td>N</td>
</tr>
<tr>
<td>4_east_4m_l4m</td>
<td>45</td>
<td>5.66</td>
<td>N</td>
</tr>
</tbody>
</table>

These results, in conjunction with Tables 5.6 and 5.7 would imply that at 45 degrees, the reliability of the result decreases sharply, with only one of four images taken at 45 degrees being identified successfully. The location of these buildings makes taking the images a significant distance away from the building far more difficult.

The 4 East test also produces an interesting result in the case of the image being taken too close to the building. In this case 2m from the building is clearly to close for the building to be identified. The importance of “distance” from the target building was highlighted in the data set revolving around Barclays. Due to it’s position, it’s viewpoint invariance was tested, not by a succession of different angles and movements left and right of the building’s normal – but by moving progressively further from the building.
Table 5.8: Results from Barclays

<table>
<thead>
<tr>
<th>Filename</th>
<th>Angle</th>
<th>Distance from building</th>
<th>Building Correctly identified</th>
</tr>
</thead>
<tbody>
<tr>
<td>barclays 2m</td>
<td>0</td>
<td>2</td>
<td>N</td>
</tr>
<tr>
<td>barclays 4m</td>
<td>0</td>
<td>4</td>
<td>N</td>
</tr>
<tr>
<td>barclays 6m</td>
<td>0</td>
<td>6</td>
<td>Y</td>
</tr>
<tr>
<td>barclays 8m</td>
<td>0</td>
<td>8</td>
<td>N</td>
</tr>
<tr>
<td>barclays 10m</td>
<td>0</td>
<td>10</td>
<td>N</td>
</tr>
</tbody>
</table>

Figure 5.22: The database image of Barclays (top) and the query images at (left to right) 2m, 4m, 6m and 8m. Only the 6m was successful, highlighting the importance of

These results, as well as those for 4 East, seem to imply that there is an optimal distance within which a successful result can be obtained. Defining the “right distance” to be away from the building is a difficult concept because of the way the database was constructed. In each case the photograph of the building was taken in such a way so as to encompass the whole building in the image. As such, each building responds differently to distance – though rules regarding angles appear broadly similar, with up to 40 degrees either side of the normal of the building with no clutter, usually being successful.

5.6.3 Illumination

In section 5.2 we attempted to assess the effect illumination had on an image by artificially lowering the contrast using a standard computer graphics package – however, this is not a good representation of taking an image in poor illumination. In order attempt to attach a metric to the degradation in natural illumination, it was decided to take photographs from the same position on a regular basis throughout the day. The times chosen were 3pm, 6pm 9pm and Midnight. The buildings chosen were Wessex House Lecture Theatre, the Chaplaincy, UBSA and 8W2.1. Figures 5.23 and 5.24 make the degradation clear.
Table 5.9: Tests with Illumination

<table>
<thead>
<tr>
<th>Building</th>
<th>Time</th>
<th>3pm</th>
<th>6pm</th>
<th>9pm</th>
<th>Midnight</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHLT</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>UBSA</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Chaplaincy</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>8W2.1</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

The success of the images at 3 and 6 pm is not surprising due to their similarity to the database images (which were originally taken at around 2pm on a reasonably bright March day). The difference between 6pm and 9pm is, however, marked (as figure 5.25 demonstrates).

This is to be expected, although is a disappointing result due to the number of precautions taken throughout the development to minimise the effect that illumination could have (Sections 5.2 particularly). The sudden drop prompted the author to attempt to find the “cut off” point more precisely, and an identical drop was seen the next day, between 8 and 9pm.

A plot of the H and S points for each of these images provides some clues as to the origin of this problem. Each building was given a unique set of points so their average position in HS space could be monitored. Each building appears to be “clustered” for the first few points – presumably these are the ones which are correctly identified. However, there are a few which are further away. These ones are, presumably, the ones that fail the test – they have reached a level of illumination where it is far more than just the V component which is affected – but H and S have moved dramatically across the space, also.

Figure 5.23: Wessex House Lecture Theatre at 6pm (right) and 9pm (left). The difference is noticeable and the program fails to cope in most cases.
The most interesting of the illumination results are those regarding the Chaplaincy. Certain images of the Chaplaincy in the early tests gained good results, and its distinctive colour (the building is almost entirely blue) meant it was amongst the more quickly identified buildings. It was because of these indications that the chaplaincy tests proved so surprising. As table 5.9 shows, the building failed to be identified at 3pm. Figure (I'MAHIPPY) shows the building at the various times.

The image earlier in the day is clearly affected dramatically affected by the shadow being cast by the library. This pushes the average colour outside the initial hashing techniques and the building is eliminated at an early stage. The techniques described to combat illumination problems are unlikely to be able to cope with contrasts of this kind, although figure 5.26 clearly shows how
considering H-S space clearly helps. This is a shame as in the areas where it is light, it is not particularly brighter than the database image, and the darker patches are not far removed in colour from those seen in the 9pm case. However, the contrast appears to be causing issues. One further attempt to solve problems of this nature would be to build on the idea of “pixel normalisation”, implemented in a similar system to this one by Cipolla and Robertson, seen in section 2.4.

5.6.4 Times

The introduction of each testing technique led to an increase in the time taken to run the program – the only part of the whole process which saved any kind of time were the hashing techniques based on colour space discussed and implemented in section 5.4.2.

However, results so far imply that while this is speeding up the process, it is having an adverse effect on the results. For this reason, it made sense to analyse exactly how much faster the building tests are thanks to the initial hashing by colour space.

The best way to do this seemed to be a straight comparison by running the program with and without the technique. The first set of results show the buildings identified with a program incorporating the technique. The times were obtained using the Matlab “profile” function which returns the time taken for the function to work, as well as the times taken for all sub-functions to run. This lets us assess what percentage of the overall running time is taken up by the process.

Table 5.10: Comparing the Colour Hashing Method

<table>
<thead>
<tr>
<th>Building</th>
<th>Profile Time with Colour Checking (seconds)</th>
<th>Profile Time without Colour Checking (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>URB</td>
<td>141</td>
<td>1059</td>
</tr>
<tr>
<td>4E</td>
<td>1034</td>
<td>2712</td>
</tr>
<tr>
<td>8 West 2.1</td>
<td>870</td>
<td>2712</td>
</tr>
</tbody>
</table>
These are just a selection of the results which show a range of times, with the slightly more distinctive buildings being identified quickly. URB is one of the few buildings to be correctly identified within the time mentioned in requirement 10 (three minutes). However, those that take longer are the ones where the building-eliminate function has constructed a significantly longer shortlist. It would seem the only way to improve this would be to decrease the thresholds, and incur the problems of a large and cumbersome database. So, how does this compare with a system in which there is no such quick-search method? The following results, again using the profile function in Matlab, show the same searches without the building-eliminate process – and compare the results with what we have seen before.

It would imply that the process is worth running due to the significant amount of time taken to test – and a careful balance should be struck between accuracy and efficiency.
6. Conclusion

6.1 The Project

In the literature review, it became apparent that there was more than one way to find features in an image and, as such, more than one way of developing a search mechanism in a system such as this. In some cases, it was simply a matter of making a decision based on the performance evidence (such as in deciding between RGB and HSV in the comparison function) but more often it is a case of successfully combining methods and using them to either eliminate buildings (the colour space hashing) or to contribute to a scoring system (edges and corners).

The idea of Computer Vision becoming a balancing act between the different information extracted from an image is not an unusual one. “Hawkeye”, Channel 4’s cricket ball tracking system, uses six cameras to trace not only the trajectory of the ball, but also the swing of the bat and the nature of the bounce of the ball to calculate its eventual path [40]. SPECS speed cameras are equipped to take both full colour and infra-red images at the same time to maximise information received [41]. In this particular project, we encountered the problem of two of the methods being in direct contrast with each other.

While the corner, edge and image comparing techniques constructed appear to be resistant to illumination problems, they are let down by the hashing techniques which, while incredibly efficient in full illumination, let the system down in poorer illumination.

These two aspects were both requirements (numbers 6 and 10, respectively) and made the efficiency versus accuracy problem very clear. As we have seen in the section 5.5, to lessen the effect of the hashing would have had implications on the rest of the project regarding its efficiency – particularly during daylight hours. Perhaps programming in Matlab has emphasised this point a little more than if it had been implemented in another language.

The altering of parameters and weightings aside, this project has certainly shown that the identification of buildings from a poor quality query image is certainly possible by the methods implemented – and in that sense it was a success. It has also highlighted that computer vision is often a balancing act in which various aspects of information implied from an image are more useful in combination with others, than when considered independently.

6.2 The Software Process

The evolutionary design model used to approach the project turned out to be a sensible method. It meant the project could be continuously adapted and with more time, more improvements could have been made. It is likely the project would have moved in the direction of another method with which to hash the database using other information quickly obtained from the images. Number of windows could be one, with obscured or out-of-range photographs having to be
dealt with. In order for substantial progress to be made, we would almost
certainly need to move to a more efficient language, such as C, as testing
poorer quality images without the hashing took an impractical amount of time.
Were such a move to have the required impact on the time taken for the
program to run, the possibility for further research into greater accuracy and
performance in poorer illumination could be looked into even further.

6.3 Future work

This project clearly has a large amount of room for expansion, aside from
implementing further techniques to make it more reliable or by expanding the
database. The project is easily adaptable to other areas. By simply replacing the
Matlab structure “campus” throughout the code, the system could be
implemented anywhere where there are a reasonably distinctive set of buildings
or planar objects.

One extension could be a system which is capable of learning from its mistakes,
like the Gracenote CDDDB system, which attempts to identify the track list of a
CD entered into a computer, based on an on-line database [42]. The system is
kept up to date not only by the staff, but also by the contributions of the users,
who are asked to submit track lists to more obscure CDs, or submit errors in the
database. These contributions are checked by staff for errors or multiple
submissions and then entered into the database. Such a dynamic system would
allow the system to be kept constantly up-to date.

On a large enough scale, a system of this type could be similarly maintained –
especially in urban environments where the landscape can change. Even in the
implemented example of the campus, one of the larger buildings (4 West), was
under construction when the database images were collected and, as such,
could look significantly different in a few months.

The inability to cope with frequently changing landscapes is amongst the main
obstacles in implementing the systems in “non-urban” environments. Countryside regions change more dramatically with the changing of the seasons
and would suffer from few regions being capable of being called planar. For this
reason “Urban Navigation” is a surprisingly important constraint!

This leads to another possible extension - the ability of the project to cope with
non-planar objects. Although, in an urban environment, most buildings could be
modelled as planar, most landmarks, such as statues, could not be safely
labelled as such. An interesting extension would be to see how well the methods
implemented in this project transfer to non-planar objects and how the database
would have to be modified in order to cope (for example, with objects
photographed at a range of angles).
Bibliography


2. GARMIN LTD, GPS – A User’s guide. Outdoorgear UK Ltd, 72/74 Palmerston Road, Bournemouth, Dorset, BH1 4JT: Outdoorgear.co.uk. Available from: http://www.outdoorgear.co.uk/gps/why_is_the_signal_wrong.htm

3. PONCE, J., LAZEBNIK, S., ROTHGANGER, F., SCHMID, C., 2004, Toward True 3D Object recognition Department of Computer Science and Beckman Institute, University of Illinois


7. FAUGERAS, O., LUONG, Q., PAPADOPOULO, T., 2001, The geometry of multiple images : the laws that govern the formation of multiple images of a scene and some of their applications, - Massachusetts, USA: The MIT Press


41. SPEED CHECK SERVICES, Speed Check Services FAQ [online]. UK: Speed Check Services Ltd. Available From: http://www.speedcheck.co.uk/FAQs.htm