Automatic People Removal from Photographs

Computer Science Tripos Part II

Fitzwilliam College

May 5, 2009
Original Aims of the Project

The original aim of this project was to write a program that, given a set of images taken from the same approximate location, can automatically extract an artificial background image by aligning the images, detecting foreground obstructions in each photo, and constructing a new image out of the remaining background regions. It should also allow a user to manually align the images, and to manually amend the anomaly selection. There were also two extensions: to add panorama stitching to the image alignment, and to add in-painting where a non-anomalous region cannot be found.

Work Completed

The program, written in C#, performs all of the above non-extension aims, except for manual image alignment. It can automatically align images with high confidence, detect occluding elements based on a background approximation, and seamlessly blend the images together, avoiding these occlusions where possible. The image feature detection and matching (for alignment) uses an external library, however everything else is code written for this project. Also, one extension has been implemented: panorama stitching.

Special Difficulties

None.

Declaration of Originality

I, Lech Świrski of Fitzwilliam College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed

Date
Contents

1 Introduction 1

1.1 Motivation .................................................. 1

1.1.1 Perceived Scenes .................................. 2

1.1.2 ‘Tourist Removal’ .................................. 2

1.2 Related Work ............................................ 3

1.2.1 Motion Detection .................................. 3

1.2.2 Panorama Stitching ................................. 4

1.3 The Project ............................................... 5

2 Preparation 6

2.1 Aims ....................................................... 6

2.2 Requirements Analysis ................................. 6

2.3 Preparatory Research ................................. 7

2.3.1 Homographies ................................... 7

Affine Transforms ....................................... 7

Perspective Transforms ............................... 8

2.3.2 Colour Distances ................................ 8

2.4 Software Development Process ....................... 9

2.5 Development Environment ......................... 10

2.5.1 Programming Language ......................... 10

C++ ......................................................... 10

MATLAB .................................................. 11

Java ......................................................... 11

C# .......................................................... 11

2.5.2 Integrated Development Environment ......... 12

2.5.3 Version Control ................................. 12
## 3 Implementation

3.1 Framework ................................. 14
   3.1.1 Linear Algebra ............................. 14
      Matrices and Vectors ......................... 14
      Matrix Decomposition ....................... 15
   3.1.2 Bitmap Manipulation .................... 16
      Interpolation ................................ 18
3.2 Graphical User Interface .................. 20
   3.2.1 Technology ............................... 20
   3.2.2 Interface ................................. 20
      Images ...................................... 22
   3.2.3 Masking .................................. 22
      Drawing Masks ............................. 22
      Erasing Masks .............................. 23
      Brush Cursor ............................... 24
3.3 Image Alignment ............................ 25
   3.3.1 Feature Point Detection ................. 25
   3.3.2 Pair-Wise Feature Matching .............. 25
   3.3.3 Pair-Wise Alignment ..................... 26
      RANSAC .................................... 27
   3.3.4 Global Alignment ........................ 29
      The Levenberg-Marquardt Algorithm .......... 29
   3.3.5 Manual Alignment ......................... 30
3.4 Occlusion Detection ....................... 31
   3.4.1 Colour Median ............................ 32
   3.4.2 Problems With The Median ............... 32
   3.4.3 Post-Processing ......................... 33
      Median Post-Processing .................... 33
      Mask Post-Processing ...................... 34
3.5 Selective Blending ........................ 36
Introduction

My project aims to allow a photographer to extract a static background from a set of photos of a location occluded by unwanted foreground elements, such as tourists. From a set of imperfect, partially obscured photos, it extracts the perceived background the photographer had initially wanted to capture, using a similar approach to panorama stitching. I have successfully implemented this method, and partially implemented one proposed extension: extending the image alignment code to cope with sets where not all photographs are pair-wise overlapping.

Figure 1.1: An example of background extraction, using manual selection with multi-band blending.

1.1 Motivation

The motivation behind this project is to create a virtual photograph that represents a perceived, stationary background scene. Each photograph has transient occluding elements,
such as people and cars, that a human can subconsciously ignore yet a camera innocently captures. By instead using information from several photographs, we can extract the perceived scene instead of the actual scene.

1.1.1 Perceived Scenes

It is a common problem in photography that photographs taken of a scene always present the scene objectively. The human eyes and brain have evolved in such a way that what one ‘sees’ is only a subjective interpretation of the scene, and is an amended version of the reality before them. For example, the human eye only has high-resolution, colour vision in the foveal region, whereas the majority of the retina provides low-resolution, grayscale peripheral vision—yet, the scene perceived by our conscious brain is uniformly coloured and sharp. Furthermore, we build up an image of the world around us based on input over time; when the view of a building is partially blocked by a person walking in front of it, we have no problem ‘seeing’ the parts of the building behind the person even though we cannot see them at that exact moment, as we had seen it before. However, when a photograph is taken, it impartially photographs the scene as it is, with all occlusions as they are.

More often than not, a casual photographer will want to capture his perceived scene, rather than the objective one. It is clear that the image he sees is one post-processed by his brain using information collected over time, therefore presumably it should be possible to construct a virtual image approximating the perceived scene using several photographs of the same location. In many cases, when one wants to take a photograph of a stationary object, there will invariably be a moving, anomalous object—a bird, a person, a car—in the photograph which blocks an element in the photograph. More often than not, one can take several photographs of a single scene, where each part of the background is visible in the majority of the photographs, and yet anomalies in each mean that the background is not wholly visible in any of them.

Under this assumption, it should be possible to automatically construct a virtual photograph of the static background, as originally perceived by the photographer, by analysing the constants between images.

1.1.2 ‘Tourist Removal’

The original and primary motivation behind this project is to remove tourists from photographs of tourist destinations. The major problem with photographing popular tourist destinations is that it is very difficult to take a photograph without strangers in it. This is a problem both for casual photographers (as in figure 1.2) and for professionals, where
the latter may need to take a photograph without people, for example for some publication. A common approach used by professionals is to take photographs with extremely long exposure, effectively performing an integral of the scene over time, where moving foreground occlusions have a negligible impact on the overall camera input. This has two problems:

1. It requires a tripod to keep the image steady, as the slightest movement would introduce blurring. Few casual photographers would carry a tripod around with them.

2. It requires a lens with an extremely small aperture: the long exposure will mean that the lens would have to strongly limit the amount of light reaching the sensor, otherwise the photograph would be highly overexposed. This is impossible for small, cheap cameras where lenses are not interchangeable, and even when available for more professional cameras, lenses with such low aperture are very expensive.

Therefore, the motivation of this project is to allow a casual photographer to take several photographs of a scene using a hand-held camera, and then have them digitally combined to create a composite image that is an approximation to the perceived scene, without tourists in the way.

### 1.2 Related Work

#### 1.2.1 Motion Detection

The work of a motion detector is similar to background extraction in that it requires the separation of a static background from its foreground. Motion detectors can use back-
ground subtraction, which uses a fairly simple (and adapting) estimation of the background of a scene to detect anomalous objects, and hence movement. An example is Monnet et al. (2003). This is commonly done by using information from a video collected over time—as a brief example, Monnet et al. (2003) works by using a function of the weighted average of previous video frames to create a prediction for the next; this prediction is essentially an estimate of the background. Differences to this estimate are then considered as moving elements. However, this background is only a rough estimate, and is only calculated so that it can be discarded and the anomalies presented. Background extraction performs the opposite.

1.2.2 Panorama Stitching

![Panorama stitching](image)

**Figure 1.3:** Panorama stitching takes several photographs of a scene, and composites them into a single, panoramic image.

Panorama stitching is the process where several images of a large scene, where each photograph only captures a part of the scene, are combined into one composite image presenting the scene in its entirety. Many commercial applications, including camera software, have a facility for panorama stitching, and it has an extensive research literature (Szeliski, 2006).

Panorama stitching is split into two parts: alignment and blending of images. The alignment is commonly done using some form of feature detection to detect matching features
between pairs of images, and then finding transformations between images to align them. Features in older papers were commonly image patches around Harris corners (Harris, 1993); a good example is Zoghlami et al. (1997). Here, image patches around ‘corner’ elements, where two edges intersect, are compared to find matches. However, this is not particularly robust to affine transformations (such as rotation or scaling), so recent research, such as Brown and Lowe (2003) and its follow-up Brown and Lowe (2007), has begun to use invariant feature descriptors, in particular the scale invariant feature transform (SIFT, Lowe, 2004).

Blending in panorama stitching varies, but the common principle is that each pixel in the final image is a weighted average of the corresponding pixels in the aligned images, usually weighted by their distance to the centre of their respective image (Szeliski, 2006). Brown and Lowe (2007) uses a more sophisticated, multi-band blending method (see section 3.5.2), however the principle is the same.

Panorama stitching is similar to background extraction in its logic of creation of a single virtual scene from several photographs presenting incomplete information. It handles image alignment, which is a requirement for hand-held camera photographs as described above. The blending stage also handles merging multiple overlapping images, using some form of weight for the images’ values to decide the pixel, which is similar to a background extractor weighing inputs from images to remove unwanted elements.

1.3 The Project

My project’s main aim is to explore and adapt the work presented in the Brown and Lowe (2007) panorama stitcher into a background extractor, using the methods as described for panorama stitching to be able to align images, and to create a composite image where the selected regions in images are ignored in the final composition. Furthermore, I will use the principle of estimated background in motion detection to detect anomalous elements between images, so that this information can be used in the background extraction. All these steps should be fully automatic. The user should also be able to manually amend the anomaly selection process, for example if one of the automatically detected ‘anomalies’ is in fact a posing family member.
Preparation

This chapter discusses the preparation performed before beginning the implementation of this project. Any project of non-trivial size requires a preparatory stage before work on the actual project can commence, and this project is no exception. This chapter outlines this preparation, including an analysis of the requirements, the proposed stages of the implementation, the choice of programming language and development environment, and the preparatory research performed.

2.1 Aims

The main aims of this project are summarised in table 2.1.

<table>
<thead>
<tr>
<th>Aim</th>
<th>Priority</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implement GUI for loading and painting on images</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Implement or find library for feature detection</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>Implement automatic image alignment</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>Implement anomaly detection</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Implement image blending</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>Implement panorama stitching</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Explore possibility of in-painting</td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>

*Table 2.1: Summary of project aims.*

2.2 Requirements Analysis

It was decided near the beginning of the project that the requirements of the final project are that it is able to:

1. Automatically align images
2. Automatically select areas which should be removed to expose the background,
3. Allow the user to manually select or deselect areas to be removed, and
4. Seamlessly stitch the desired areas together to show only the background.
Performing these tasks automatically with no assumptions about the images is infeasible, therefore I require my project to be able to perform under the following assumptions:

1. **No parallax.** The camera does not move except for rotations and zooming.

2. **Constant camera parameters.** Aperture, exposure, white balance, ISO, etc. are constant between photographs.

3. **Significant overlap.** All photographs overlap significantly.

4. **Majority background.** Each part of the background is visible in a majority of photographs, even if the background is not entirely visible in any of them.

Also, two extensions were proposed, to decrease the need for the above assumptions.

1. **Full panorama stitching:** this eliminates the need for significant overlap.

2. **In-painting:** this eliminates the need for a majority background, but could require an additional, manual stage to select regions to be in-painted.

### 2.3 Preparatory Research

#### 2.3.1 Homographies

In this project, one of the assumptions is that the camera can only rotate. This means that all the images can be related by a 2D perspective transformation, referred to as a planar homography. This section summarises the properties of homographies that were researched for this project. The main source of research for this area was Hartley and Zisserman (2004).

**Affine Transforms**

Transforms, in general, are functions that map an \((x, y)\) coordinate pair into a new \((x', y')\) coordinate pair. Another way of seeing this is that they map points in one image to points in another. The affine transforms are rotation, scaling, shearing and translation, and any combination of these. These have the interesting property that, except for translation, they can be represented as a 2D matrix multiplication of a vector representing the coordinates:

\[
\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}
\]  

(2.1)
To represent translation, we use homogeneous coordinates. This is a three dimensional space of coordinate triplets \((x, y, w)\). We map these into the 2D plane with the mapping \((x, y, w) \mapsto \left(\frac{x}{w}, \frac{y}{w}\right)\), and we get homogeneous co-ordinates from an \((x, y)\) pair with the natural inverse mapping \((x, y) \mapsto (x, y, 1)\).

With homogeneous coordinates, we can now represent affine transforms with a 3D matrix:

\[
\begin{pmatrix}
  x' \\
  y' \\
  w
\end{pmatrix} = \begin{pmatrix}
  a_{11} & a_{21} & a_{31} \\
  a_{12} & a_{22} & a_{32} \\
  0 & 0 & 1
\end{pmatrix} \begin{pmatrix}
  x \\
  y \\
  w
\end{pmatrix}
\]

(2.2)

where \(a_{31}\) and \(a_{32}\) are a translation in \(x\) and \(y\). Notice that \(w\) does not change; indeed, in affine transforms, \(w\) remains 1. Also, this type of matrix is closed under multiplication, which means that any combination of affine transforms is still an affine transform.

**Perspective Transforms**

The matrix in equation 2.2 describes all possible affine transforms, but does not describe all possible \(3 \times 3\) matrices. Therefore, there exists a class of transforms more powerful than affine transforms, which uses the full \(3 \times 3\) matrix. This class is the perspective transforms, which we call homographies.

Another way to describe homographies is that they relate two images taken by a camera. Assuming a pinhole camera model, homographies can relate any two images taken by the camera where its focal point has not been translated; this means that only rotation and zooming is allowed.

### 2.3.2 Colour Distances

The concept of colour distance is important in this project. I wish to be able to compare the distances of pixels, to detect outliers, and this can best be done using the distance between the pixels’ colour values.

Colour is, by the build of the human cornea, a three-dimensional space. Therefore, the natural definition of the distance between two colours is the Euclidean distance between them in this three-dimensional space.

\[
d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}
\]

(2.3)

where \(x_i, y_i\) and \(z_i\) are the values of image \(i\) in the \(x, y\) and \(z\) dimension, respectively. The values of a colour in each of these dimensions will depend on the colour space used, for example the RGB colour space will have \(x, y\) and \(z\) equal to \(r, g\) and \(b\), respectively.
We also want the distance to be perceptually valid; that is, for any two pairs of colours that are the same distance apart, we also want them to be qualified visually as equally different. The Euclidean distance in the standard RGB colour space is therefore not a good colour distance, as the RGB space is not perceptually uniform. We therefore seek a perceptually uniform space, such as CIELAB or CIELUV (Reinhard et al., 2008). In these two spaces, the colour distance performs better; extensive testing has shown that a modified lightness curve makes them perform even better (Granger, 1994). However, converting the whole image to CIELAB or CIELUV is expensive.

In the end, I decided to use a computationally efficient approximation to Euclidean distance in CIELAB (with modified lightness curve) using RGB values, as found experimentally in Riemersma (2001).

\[
\bar{r} = \frac{r_1 + r_2}{2} \quad (2.4) \\
\Delta r = r_1 - r_2 \quad (2.5) \\
\Delta g = g_1 - g_2 \quad (2.6) \\
\Delta b = b_1 - b_2 \quad (2.7)
\]

\[
d = \sqrt{\left(2 + \frac{\bar{r}}{256}\right) \Delta r^2 + 4\Delta g^2 + \left(2 + \frac{255 - \bar{r}}{256}\right) \Delta b^2} \quad (2.8)
\]

### 2.4 Software Development Process

A project with a one-person team has a unique dynamic not present in any larger team, where it is the case that one person knows the workings of the entire project, and does all of the specification, development and verification. Furthermore, a precise specification could not have been available at the start of the project, as it was exploratory by design.
Therefore, an iterative development process was both appropriate and necessary. However, the ‘vague’ specification of the project divided it into partially separable modules, which could each be precisely specified, developed and verified semi-independently.

I therefore decided to use an incremental evolutionary prototyping development process, where the specification, development and verification of each module is performed separately. The modules are developed in an evolutionary way, where there is a feedback process of the specification affecting the development, and the development and verification amending the specification. The module prototypes are then merged into the overall design. The merging will show deficiencies in the modules’ specifications, which will be amended, and this process will be iterated. This is summarised in figure 2.2.

![Overall specification outline](image)

Figure 2.2: An incremental evolutionary prototyping development process.

### 2.5 Development Environment

#### 2.5.1 Programming Language

The project is implemented in C# 3, a Microsoft language in the .NET framework that is in many regards similar to Java. However, this was not the only language available to me. This section describes several alternatives, and why I rejected them, as well as describing my reasons for picking C#.

**C++**

C++ has the speed advantage over any language running in a virtual machine; however, advancements in Just-In-Time (JIT) compilation have narrowed this performance gap,
therefore this is not enough of a reason to pick it. It has an incredibly large codebase, however, the C++ standard library is relatively small. This means that I would have to find many non-standard libraries for simple tasks, such as loading images, which would significantly slow down development.

Also, it is difficult to write GUIs in C++, and memory management is difficult in large projects. The above disadvantages, combined with my relatively little experience with C++, made me consider other possibilities.

**MATLAB**

MATLAB is an excellent language for linear algebra; however, I have had little experience with it, and I believe that the cost of learning to program and debug MATLAB—in particular for image manipulation—combined with the risk of trying this language when it may not be suitable, made me decide to use another language.

**Java**

Java is a very easy language to write in, and shares many similarities with C#. However, for the ease of the programmer, it misses several language elements that are present in C#: examples include non-virtual functions, operator overloading, and lambda expressions. On the upside, Swing is much easier to write GUIs in than C++’s MFC (a standard C++ wrapper for the Windows API), and memory management is not a problem with Java’s garbage collector. Java is a good language choice, and I have had a lot of previous experience coding in it; if not for C#, it would most likely have been my language of choice.

**C#**

C# is a fairly new language, comparable to Java in many ways: it runs in a virtual machine, it has a garbage collector, it has a large standard library and it has a simple GUI writing technology. However, it also has several advantages over Java. There are language features that Java doesn’t have, such as operator overloading, lambda expressions, property fields with transparent getter/setter methods, and the ability to run ‘unsafe’ code, where one can access memory directly with pointers. Also, WPF (Windows Presentation Foundation) is a new, Swing-like technology for writing GUIs, but limited experience has made me believe that it is easier to use than Swing.

Although C#’s standard 2D graphics libraries only support affine transforms, Java has the same problem, so this is not a disadvantage. The above advantages, coupled with a standard IDE (Visual Studio .NET) that I prefer to Java’s standard Eclipse or Netbeans,
and with a large amount of past experience with C#, I decided to choose it as the language used for my project.

2.5.2 Integrated Development Environment

Development was carried out in Microsoft Visual Studio .NET 2008, which is the standard and de facto only IDE for C# coding used by professionals. It provides an excellent development environment, with code completion, easy code traversal, and a very good debugger.

2.5.3 Version Control

The project code was hosted in a Subversion repository on the University of Cambridge Student Run Computing Facility servers. This provided two advantages: version control allowed me to revert to a previous version of the project if something went wrong with my local copy; and this provided an external backup in case my local copy got lost.

For added security, I also wrote a batch script which periodically \texttt{tar}s and compresses my project, and uploads it to an external server as a backup. Indeed, this backup system was invaluable in recovering from a fatal hardware failure during the last few weeks of the project.
Implementation

This chapter describes my project’s implementation. This can be divided into three main sections: framework, graphical user interface and image processing. The image processing section is in turn split into three sections: alignment, occlusion detection and selective blending. These form a three-stage image processing pipeline.

1. **Framework**
   
   This has two main sections:
   
   (a) **Linear Algebra**
       
       Classes for matrices and vectors, and functions that operate on them (such as matrix decomposition).
   
   (b) **Transformable Bitmaps**
       
       Bitmap operations necessary to transform a bitmap by arbitrary projective transforms, and wrapper classes for these operations.

2. **GUI**

   Describes the front-end of the project which loads and display images, displays detected anomalies, allows a user to make modifications to the anomalies, and can output a final image.

3. **Alignment**

   Describes the feature-detection–based alignment stage for aligning a set of images.

4. **Occlusion Detection**

   Explains the automatic masking of unwanted elements using an estimated background, and the post-processing of these masks.

5. **Selective Blending**

   Extends selective blending, in particular multi-band–blending, for multiple variably masked images, to combine them into the final result.
3.1 Framework

This section explains the core components of this project, which is linear algebra and bitmap manipulation. The first, linear algebra section explains the implementation of matrices and vectors, and the linear algebra methods operating on them. The linear algebra methods focus in particular on the decompositions used in the alignment (see section 3.3). The second, bitmap manipulation section describes the interfaces for arbitrary transformable objects, followed by the bitmap implementation of these interfaces resulting in transformable bitmaps.

3.1.1 Linear Algebra

It was realised early on that linear algebra would be a critical element of the project. $3 \times 3$ matrices are necessary for projective transforms, and matrix decomposition is a good way to find a homography between pairs of points, which is necessary for image alignment. There are no standard, flexible linear algebra routines or classes in the .NET framework, so I decided to implement them myself to have full control over their internals.

Matrices and Vectors

The matrix and vector classes in this project are designed to be a fully flexible implementations of the mathematical concept. Early on, I decided that vectors should be distinct from matrices, instead of a single row/column matrix, with no concept of horizontal or vertical vectors. This is because mathematically, they are related but distinct concepts, and could lead to confusion in use. This is unlike MATLAB, which can use scalars, vectors and matrices interchangeably. However, the design of the two classes is similar.

The matrix class is a wrapper for a two dimensional array of double-precision floating-point values\(^1\), whose dimensions are decided in the matrix’s constructor. The constructor is also overloaded to allow one to assign individual values to the matrix, assign values to the diagonal of the matrix, fill the matrix with one value, or fill the matrix according to a function. The latter is an interesting use of C#’s lambda expressions (as in code 3.1). The matrix and vector classes also use a large amount of operator overloading, to simplify their usage. Overloading based on the arguments allows us to have the same syntax for multiplying two matrices, as well as pre– or post–multiplying a matrix by a vector.

\(^1\)Double-precision floating-point values are used to maximise accuracy, as matrices are expected to be small enough for this to have little memory impact.
public Matrix(int rows, int columns, Func<int, int, double> filler)
{
    data = new double[rows, columns];
    for (int i = 0; i < rows; i++)
        for (int j = 0; j < columns; j++)
            data[i, j] = filler(i, j);
}
...
Matrix m = new Matrix(5, 5, (x, y) => x == y ? 1 : 0);

Code 3.1: Filling a matrix using a lambda expression to create the $5 \times 5$ identity matrix.

Matrix Decomposition

Matrix decomposition is a form of matrix analysis which represents a single matrix as the product of several other matrices. Decompositions are often useful when they guarantee the properties of the decomposition matrices, as they can demonstrate properties of the original matrix that are not immediately obvious. There are two types of decomposition implemented in my linear algebra classes: singular value decomposition (SVD) and LUP decomposition.

**Singular value decomposition** is the decomposition of an $m \times n$ matrix $M$ as:

$$M = U\Sigma V^T,$$

(3.1)

where $U$ and $V$ are unitary $m \times m$ and $n \times n$ matrices respectively, $V^T$ is the transpose\(^2\) of $V$, and $\Sigma$ is an $m \times n$ matrix with values only along its diagonal.

SVD decomposition is used to find the homography of four points (see section 3.3 for usage). My implementation is based on Wilkinson and Reinsch (1971), the same as used in MATLAB. I found an implementation in a Java linear algebra package, JAMA (Hicklin et al., 2000), based on a Fortran library, LAPACK (Demmel, 1989). For my project, I rewrote this implementation in C#, adapting it to my own matrix classes\(^3\).

**LUP decomposition** is a form of LU decomposition, generalised to rectangular matrices. For a square matrix $M$, the LU decomposition is:

$$M = LU,$$

(3.2)

where $L$ and $U$ are lower and upper triangular matrices—they have only zeroes above and below the diagonal, respectively. The LUP decomposition adds a pivot: a square permutation matrix $P$, which is a matrix of zeroes and ones such that each row and

\(^2\)Mathematically, we need the conjugate transpose here. However, as I deal only with real numbers, the transpose is sufficient.

\(^3\)In fact, the JAMA SVD is buggy for some rectangular matrices, so I've adapted a slightly modified version from http://cio.nist.gov/esd/emaillist/lists/jama/msg01430.html.
column has exactly one 1; this results in a permutation of the rows of a matrix it pre-
multiplies, so that:

\[ M = LUP. \]  

(3.3)

The LU decomposition only exists for square matrices, and only under certain conditions (Okunev and Johnson, 2005). However, an LUP decomposition exists for every matrix, both square and rectangular. LUP decomposition is a modified form of Gaussian elimination, which is a method for solving systems of linear equations, and is useful in solving linear equations of the form \( P^{-1}Mx = LUx = b \) for \( x \). This is done by solving the equation separately for the two triangular matrices, with \( Ly = b \) and \( Ux = y \), each of which can be done simply using standard Gaussian elimination.

The LU decomposition is used to find the inverse of matrices in a computationally efficient way. My implementation was again based on Wilkinson and Reinsch (1971), using code from JAMA translated to C#.

### 3.1.2 Bitmap Manipulation

The project requirements state clearly that images have to be able to be transformed by applying an arbitrary perspective transform, which is a transform by an arbitrary \( 3 \times 3 \) matrix. This is a problem in .NET, as there only exist facilities for affine transforms (see section 2.3.1). Therefore, I had to write my own implementation of arbitrary \( 3 \times 3 \) matrix transformations, as outlined in figure 3.1.

![Figure 3.1: A succinct class diagram of the bitmap manipulation classes.](image)

Since I wanted other things—for example, the positions of feature points—to be able to transform alongside an image, I first created an abstract \textit{Transformable} class, which holds a field for the matrix, and a quadrilateral object for the outline of the transformed object. There are also functions to update the outline based on the current matrix and the original bounds, and a function called by setting the matrix which starts a new thread to update
the `Transformable`. The update code itself is a pure virtual function, to be implemented by subclasses.

There are three direct subclasses of `Transformable` (figure 3.1). Two are simple—there is `MaskableImage`, a wrapper for several other `Transformable` subclasses which is used in the GUI (discussed in section 3.2.2); and `AlignmentMask`, which displays feature points on an image. These have trivial implementations of the update code: `MaskableImage` simply forwards the matrix on to its children; and `AlignmentMask` sets the \((x,y)\) co-ordinates of objects representing feature points.

The third subclass is `AbstractTransformableImage`. There are two logical types of transformable image, which have a large set of similar functionality, but also significant differences. The first type is images which have an original, from which the transformed version can be losslessly calculated on each update. In this program, the input images are of this type, and are encapsulated in the `TransformableImage` class. The second type is images which change between transforms, and therefore have to have their new, updated version calculated from their previous. The `PaintCanvas` class used to select objects with a user-modifiable red mask is of this second type. The superset of the functionality of these two types is held in `AbstractTransformableImage`.

In the first case, we always want to keep the original and transform a copy of it. Therefore, the transform matrix applied to the original image is simply the new transform matrix of the class.

\[
\text{Transform Matrix} = M_i
\]  

(3.4)

In the second case, we want to transform the image back from a previously transformed version, and then transform this using the new matrix. By associativity of matrix multiplication, this is equivalent to transforming by the product of the new matrix, and the inverse of the previous matrix. The transform matrix applied to the image therefore takes the form:

\[
\text{Transform Matrix} = M_iM_{i-1}^{-1}
\]  

(3.5)

The majority of shared bitmap transformation functionality is the transforming of one bitmap into another, using a given matrix. This is done by creating a new bitmap, which is the size of the bounding box of the transformed outline of the old bitmap. Then, for each pixel in the new bitmap, we find the corresponding point in the original using the inverse of the matrix provided. We can now set the value of the new pixel by interpolating the values of the pixels around this point.
Interpolation

In simple, nearest-neighbour interpolation, we simply take the value of the pixel nearest to the point (figure 3.2). This results in aliasing in the resulting image, in particular jagged edges when up-scaling (figure 3.3). A more aesthetically pleasing method is bilinear interpolation, where the value of the new pixel is a linear interpolation of the four pixels around the corresponding point in the old image (figure 3.6). This results in a much smoother new image. An even better image can be obtained with bicubic interpolation, which does a linear interpolation of the 16 pixels around the point, though this is slower. The type of interpolation can be specified as a property of AbstractTransformableImage.

---

4In the optimised code submitted, the AbstractTransformableImage code is mostly omitted, as the generalised functionality it provided was found to be too costly in terms of memory usage.
Figure 3.4: The same image as figure 3.2, but transformed with bilinear interpolation.

Figure 3.5: A close up of figure 3.4. Notice that the previously jagged edges are now smoother.

Figure 3.6: Bilinear interpolation. A pixel from the transformed image is projected onto the source image $I$ as the point $p$. The value of this pixel is then a linear interpolation of the four pixels in $I$, weighted by their distance to $p$. 

$$p = stI_{x,y} + (1-s)tI_{x+1,y} + s(1-t)I_{x,y+1} + (1-s)(1-t)I_{x+1,y+1}$$ (3.6)
3.2 Graphical User Interface

This section describes the graphical user interface (GUI) of the project. It explains the technology used, the interface details, and some ways in which performance was improved through use of the graphical processing unit (GPU).

3.2.1 Technology

The GUI was created using the Windows Presentation Foundation (WPF) technology, a new graphical API in the .NET framework that is being pushed as a successor to the current standard Windows Forms, itself a C# replacement for the C++–based MFC. I decided to use WPF for two reasons. Firstly, I had never used either WPF or Winforms, therefore it was sensible to pick the newer of the two. Secondly, WPF offers a wide variety of 2D graphics functionality that Winforms do not, therefore it was the natural choice.

Unlike Java’s Swing, where a UI has to be written in Java, WPF allows one to specify a layout using XAML, an XML-based language. This allows a more natural, nested approach to UI coding, where the layout hierarchy is immediately visible. Coding Swing, on the other hand, requires an imperative approach, where is is not immediately obvious which components belong where. There exist What-You-See-Is-What-You-Get (WYSIWYG) editors for both Swing and WPF which decrease the importance of this difference, however it is still common to want to edit the GUI code by hand. Code 3.2 is an example of XAML code for a button with an icon and text.

```
<ToggleButton x:Name="btnBrush">
    <StackPanel Orientation="Horizontal">
        <Image Source="images/silkicons/paintbrush.png" Width="16" />  
        <TextBlock Text="Brush" />
    </StackPanel>
</ToggleButton>
```

*Code 3.2: An example of a button coded using XAML.*

3.2.2 Interface

This section describes the implementation and reasoning behind the user interface. The interface in this project was designed around allowing the user to easily import sets of images, switch between them, and to select anomalies. To select anomalies, I chose to give each picture a user modifiable overlay, which is a red brush that they can use to ‘paint’ over objects they want removed (figure 3.7). There is a Photoshop-like toolbar to control this brush, including switching between brush and eraser, and control over the brush’s
Figure 3.7: The project GUI, with an occluding foreground element (a cyclist) marked with a red mask. The original photo is visible in the top right corner of the image.

Figure 3.8: Comparison of different brush settings, from ‘soft’ to ‘hard’.
size and ‘softness’ (figure 3.8), along with keyboard shortcuts to control these parameters. Also, the user can zoom in and out, and move the image canvas around.

Images

The images loaded into the document cannot simply be displayed as standard WPF image objects, as they will need to display additional data, namely the red mask and feature points for alignment. Also, the image will have a perspective transform applied to it, which is not naturally supported by the WPF image object. Therefore, image files are loaded into my MaskableImage class. This is a wrapper for three objects: one each for the alignment points, the mask, and the image itself. The MaskableImage is a subclass of the WPF UIElement class, which means that instances of it can be children of GUI elements.

3.2.3 Masking

Allowing a user to draw his own mask was a challenging element of the GUI programming. The three biggest problems were soft brushes, the eraser, and the cursor; I will describe the problem and the solution to each of these.

Drawing Masks

The naïve way of implementing a hard-edged mouse-controlled brush would be to draw a circle on some canvas of the same radius and colour as the brush every time the mouse moves. This works well, and it would seem that modifying this for a soft-edged brush would be a case of changing the colour of the circle to be a radial gradient that turns transparent at the edges. However, this results in an unexpected hard edge when drawing, due to slightly opaque circles overlapping (figure 3.9).

Figure 3.9: An illustration of the problem with naïve soft-edge brush rendering. The top line uses the naïve rendering, and has a harder edge along its length than at the caps. The bottom line has correct rendering, and has an even edge all around.
My solution to this problem is, when the user presses their mouse button, to draw a hard-edged multi-segment line, on top of and separate from the drawing canvas, whose corners correspond to mouse positions, every time the mouse moves. This line is of the same colour as the brush, and its width is a function of the radius of the brush and its hardness. The rendering of this line is passed through a Gaussian blur filter, whose sigma value is a function of the size and softness of the brush. This gives the appearance of a soft-edged line, and does not affect the actual drawing canvas. When the mouse is released, this blurred line is drawn onto it (figure 3.10). This process is transparent to the user, and gives the effect of drawing on the canvas with a soft-edged brush.

![Gaussian blur](image)

**Figure 3.10:** Rendering stack of a soft-edged line. When the mouse is pressed and dragged, an path is drawn independently of the previous paths. This new path is drawn with a hard edge, and is blurred as necessary. When the user releases the mouse button, this path is merged with the previous paths.

**Erasing Masks**

The eraser was also a problem. A standard way of implementing an eraser is to draw with the background colour – the problem was that the mask had two possible colours: red and transparent. Drawing with red can be done as described above, however drawing with ‘transparent’ will have no effect. In my first implementation, the code was very complicated, using opacity masks to ‘erase’ the colour, and setting individual pixels’ alpha values in a loop, which resulted in unmanageable code and very slow rendering.

In the current implementation, I changed the logic of the mask. Instead of red and transparent, the mask is actually white on black. Then, the whole maskable object is passed through a filter, which modifies the rendered result on the GPU, changes the alpha value
of each pixel to its red component, and its RGB components to red. Now the brush draws with white, the eraser draws with black, and the user sees a red-and-transparent image.

**Brush Cursor**

The cursor was the third problem. I wanted to replace the cursor with a circle of the same radius as the brush. The simple way to do this was to add a WPF circle object to the mask, which moves with the cursor, however minor delays between the mouse’s movement and the circle’s resulted in a ‘sluggish’ experience. Therefore, the program creates a bitmap of a circle of appropriate size, updating it when the brush size is changed. This bitmap is encoded as a **PNG** file, ‘saved’ to memory, and loaded it back in using the graphics device interface (GDI). GDI calls are then used convert this into a cursor. This hack is necessary as communication between WPF bitmaps and the GDI (for cursor creation) is problematic.
3.3 Image Alignment

The automatic image alignment in the project is based on a feature-based aligner. Feature-based alignment is a four stage process:

1. Feature points are found on each image individually,
2. Corresponding pairs of matching feature points are found between each pair of images,
3. Pair-wise alignments are found between images using point correspondences, and
4. Images are globally aligned using pair-wise homographies.

I describe each stage individually.

3.3.1 Feature Point Detection

The detection of repeatedly identifiable elements in an image, for example for feature matching or motion tracking, is a well researched field. The traditional feature detection method used to be finding image patches around Harris corners (Harris, 1993); however, these patches are not invariant under affine operations, such as rotation and scaling. Recent work has resulted in the SIFT algorithm (Lowe, 2004), which is invariant under similarity transforms (translation, scaling and rotation), and partially invariant under affine transforms. In my project, I assume projective transforms between homographies, therefore it would seem that SIFT’s invariance under affine transforms is too weak a property. However, for small changes in image position, affine transforms are good approximations of perspective transforms, therefore the use of SIFT is justified (Brown and Lowe, 2007). I give an overview of SIFT in appendix A.

The implementation of SIFT used in my project is not my own, but is a C# implementation called libSIFT (Nowozin, 2005). This is because SIFT is, in general, a solved problem, however it is tricky to implement and I estimate that it would have taken me several weeks to have it working correctly. Using a popular library ensured that the code is correct, and let me concentrate on other problems.

3.3.2 Pair-Wise Feature Matching

To match pairs of points, we want to find their nearest neighbours, using the Euclidean distance in their 128-dimensional feature description space. An efficient way to find nearest neighbours is to use a kd-tree (Friedman et al., 1977). A kd-tree is an multi-dimensional,
axis-aligned binary space partition tree, which recursively partitions the feature point space on the axis with least variance⁵ (figure 3.11). To speed this up, the nearest neighbour is estimated using a Best Bin First (BBF) search (Arya et al., 1998). This uses a priority queue, where the closest bins in the feature space are searched first, and there is a cut-off after a certain amount of elements in the queue. Although theoretically we need to search all the bins exhaustively to find the true nearest neighbour, in practice searching the closest bins gives a good enough result.

For this feature matching, I again use libsift. Since libsift provides this functionality, it is sensible to use it. Unfortunately, even with BBF, the nearest-neighbour selection is very slow for a large number of feature points. A recent publication (Muja and Lowe, 2009) suggests that better performance can be achieved, however this was published too late to be included in this project.

### 3.3.3 Pair-Wise Alignment

I assume that the photographs are related by a homography—that is, a projective transformation. This assumes that the camera motion is only rotation and scaling⁶, without translation; i.e. that the focal point of the camera did not move. This is to avoid the problem of parallax, where the foreground apparently moves more than the background; to align two such images, one would have to be able to individually align the depth layers, and a simple projective transform would not suffice. This was decided early on to be outside of the scope of this project.

Therefore, we want to use the pair-wise point correspondences detected above to find a homography between each pair of images. A homography can be represented by a $3 \times 3$ matrix, which ostensibly has nine degrees of freedom. However, since we are using homogeneous coordinates (see section 2.3.1), a homography is equivalent to any non-zero

---

⁵In practice, it simply cycles through the axes used.

⁶Scaling in this case is change in ‘zoom’, rather than moving forwards and backwards.
scalar multiple of itself, as

$$(x, y, w) \equiv (\lambda x, \lambda y, \lambda w) \quad \lambda \neq 0$$

(3.7)

and thus

$$H \begin{pmatrix} x \\ y \\ w \end{pmatrix} \equiv H \begin{pmatrix} \lambda x \\ \lambda y \\ \lambda w \end{pmatrix} = (\lambda H) \begin{pmatrix} x \\ y \\ w \end{pmatrix} \quad \lambda \neq 0.$$  

(3.8)

Therefore, a homography has eight degrees of freedom, which means we need eight (or more) inputs to uniquely identify it. These eight inputs come from four point correspondences, as each point correspondences provides two constraints.

To calculate the homography, we use a direct linear transformation (Hartley and Zisserman, 2004). This transforms the problem

$$Hx = y,$$  

(3.9)

where $H$ is unknown, into

$$Ba = 0,$$  

(3.10)

where $a$ is unknown. We can solve this equation using singular value decomposition. Since $Ba = 0$, this means that $a$ lies in the right null space of $B$. The right null space vector is the right singular vector of $B$, which is simply the right-most column of one of the matrices returned by the SVD. In fact, we use a slightly modified DLT, which normalises the point correspondences before applying the DLT above. According to Hartley and Zisserman (2004), this decreases the effect of noise on the output.

**RANSAC**

With more than four points, the DLT will find a least-squares error homography matching the points. This is a problem if there are false matches, as these outliers will significantly skew the homography from the correct matrix. Therefore, an approach has to be taken which ignores these outliers, yet still provides a best fit for the inliers. This is done by the RANSAC algorithm (Fischler and Bolles, 1981).

RANSAC is an acronym for RANdom SAmple Consensus. The algorithm probabilistically finds a best fit, ignoring the outliers, by iteratively selecting several random data points as assumed inliers, finding other points which are inliers within a certain tolerance, and calculating the error with just these inliers. After a certain number of iterations, the model with least error is assumed to be correct (see code 3.3).

In my RANSAC code—originally from libsift, but heavily modified by me, in particular
input data, model, n, maxerr, d;

best_model = new model(totalerror = positive infinity)
while(k < max_iterations)
{
    // Calculate a model from n random points
    parameter_points = data[n random points]
    possible_model = new model(possible_inliers)
    consensus_set = parameter_points;

    // Find all inlier for this model, whose error is less than a threshold
    foreach (point in data)
    {
        if (possible_model.error(point) < maxerr and point not in parameter_points)
            consensus_set.add(point)
    }

    if (consensus_set.count > d)
    {
        // We have found a model matching enough points, so check if it's better
        // than our current guess

        // Recalculate the model using the consensus
        possible_model = new model(consensus_set)

        // Calculate the total error of the model
        foreach (point in data)
        {
            possible_model.totalerror += possible_model.error(point)
        }

        if (possible_model.totalerror < best_model.totalerror)
            best_model = possible_model // Have found a better model, so use it
    }

    k++
}

Code 3.3: Pseudocode of the RANSAC algorithm.
to add support for non-affine transformations—the routine selects four point correspon-
dences at random, and creates a homography from those four using the DLT as above. This homography is then perturbed slightly, and a different, perturbed homography is obtained. Projecting all the points with the two homographies, the algorithm defines the maximum error for each point as the difference in its position in these two projections—this definition of error is found in Roberts (2009).

The set of inlier points matching the model is defined as those points where the distance between the measured value in the correspondence and the projected value using the (unperturbed) homography is below the calculated maximum error. If the number of inlier points is above a threshold, the model is assumed to be good, and its error is calculated as the average errors of the inliers. After a set number of iterations, the function select the best model as the one with least error, and calculates the homography based on the inlier points alone.

3.3.4 Global Alignment

With perfect computer arithmetic and feature detection above, global alignment should be a trivial case of transitivity in the homographies between images; if images A and B have homography $H_{AB}$, then it should be true that $H_{AC} = H_{BC}H_{AB}$. However, with errors in homographies and errors in floating-point arithmetic, this is rarely the case. Therefore, global alignment is a non-trivial problem.

The simplest way to solve this, which was the original implementation in this project, is to assume that all pairs of images have a homography. We can then simply align all the images to one ‘root’ image, whose transform matrix does not change. This approximation is good enough for this project, as it depends on a large amount of overlapping images, and it is not too strict a requirement that all the images have a high pair-wise overlap.

However, one of the extensions is panorama stitching, where this restriction is no longer applicable. In panorama stitching, it is normal that two images in the same panorama may not overlap, but instead are joined by images between them. Therefore, we have to align all of the images to each other, minimising the error in the final homographies with relation to their pair-wise homographies, also known as the reprojection error. This is called bundle-adjustment, and it is commonly solved by iterating using the Levenberg-Marquardt algorithm (Triggs et al., 1999).

The Levenberg-Marquardt Algorithm

The Levenberg-Marquardt algorithm, briefly, interpolates between the Gauss-Newton algorithm and the method of gradient descent. The Gauss-Newton algorithm (GNA) maximises a function by assuming that it is locally quadratic, and selects the next value based
on this assumption, using the first and second derivative to create the quadratic estimate. The method of gradient descent, on the other hand, only uses the first derivative, and assumes the function to be locally linear. The method of gradient descent converges faster that the GNA when far away from the maximum, but slower once in the vicinity. The Levenberg-Marquardt algorithm adds a dampening factor which allows it to smoothly interpolate between the GNA and gradient descent iterations, depending on estimated distance to the maximum. This gives it fast convergence both far away from the maximum, and near it.

Bundle alignment as as described in Brown and Lowe (2007), adds images to the bundle one by one, prioritising the images by the number of matches between each image and the images in the bundle. Each time it adds a new image, it minimises the mutual errors between the images in the bundle using Levenberg-Marquardt.

My project has a partial implementation of the Levenberg-Marquardt algorithm; however, time restrictions did not allow me to finish this implementation. Therefore, I adapted the Brown and Lowe (2007) bundle adjustment to only use the DLT. Similarly to Brown and Lowe (2007), my version of bundle adjustment adds images to the bundle one by one. When adding an image, it calculates the homography between the image and the bundle using the DLT, and applies this homography to the image. The images to be added are prioritised by the error in their homographies.

### 3.3.5 Manual Alignment

In the original project plan, it was my intention to allow the user to be able to manually align the images. However, the above method works above expectations, therefore I decided not to fully implement manual alignment in the GUI. Although it is still the case that feature detection and the actual alignment are two distinct stages, and that the feature detection leaves user-movable points in the GUI, it is now, by design, not possible for the user to define their own feature points. In a production version of this program, it would not be possible to move the feature points between feature detection and alignment.
3.4 Occlusion Detection

The automatic detection of foreground obstructions in this project is based on background subtracting motion detectors. The principle is that the background is estimated by calculating, for each pixel in the background, the median value of that pixel across all the images. This is, under the constraints of this project, assumed to be a good estimate of the background. For each image, we set the value of its mask’s pixels as the normalised distance of the corresponding image pixel to the median (figure 3.12).

![Figure 3.12](image)

**Figure 3.12:** Occlusion Detection. The image on the top is the original (one of a set of 20), while the image on the bottom is the mask based on the distance to the median. Whiter mask pixels indicate a greater mask value at that pixel.
3.4.1 Colour Median

In practice, the above is a simplification of what happens. It is not immediately obvious what a ‘median’ is in a three-dimensional colour space. In fact, I use the term median to mean ‘centrepont’, which is a generalisation of the median to higher dimensions. It is calculated as the point with minimum distance to all other points. The concept of distance in colour is also non-trivial, as is discussed in section 2.3.2. In short, we should not use Euclidean RGB distance as colour distance, as it is not perceptually uniform. Instead, we use an approximation to Euclidean distance in the LUV colour space.

3.4.2 Problems With The Median

The blending implementation used in the next stage of the pipeline is multi-band blending (see section 3.5.2). A simple implementation of the above detection results in a large amount of noise during the multi-band blending, as the masks are converted to bit-masks, where only the pixel with minimal mask value will be set to visible in the bitmask. With the algorithm as above, this will always favour the image which happened to be on the median. The pixel value selected as the median is likely to be unique to its image, due to minute changes of pixel value from atmospheric, camera and compression noise. Due to these minor perturbations, an area of background which is present on several images is likely to have an effectively random source image chosen out of those containing a background. Only that image will have its mask value set to 0 at that point, and will therefore be selected as the source for that pixel during blending.
The second problem is that multi-band blending gives bad results when the mask edge coincides with an edge of high contrast in the image, which is exactly what will happen when, for example, a pedestrian in dark clothes is selected as anomalous on a light grey background (figure 3.14). These two problems are decreased in two ways: median post-processing and mask post-processing.

![Figure 3.14: Artifacts created by having a mask edge coincide with an edge of high contrast in the image. On the left image, notice the mask over the person in the middle. In the right image, this person’s original location has a white outline, an effect of the coinciding edges in mask and image. Other artifacts visible are similarly caused by such masks in other images.]

### 3.4.3 Post-Processing

#### Median Post-Processing

The median post-processing is the application of a cross-bilateral filter centred on the median, an extension of a normal Gaussian blur (Paris et al., 2008). One can calculate the value of a pixel $p$ in a Gaussian blurred image as a weighted average of all the original image’s pixels, weighted by the Gaussian function applied to each pixel’s distance to $p$:

$$I'_p = \sum_{q \in S} G_\sigma (\| p - q \|) I_q$$

where $S$ is the pixel domain of $I$ (3.11)

where $G_\sigma (x)$ is the zero-centred Gaussian function with variance $\sigma^2$. One can extend this to a bilateral filter by introducing a second Gaussian function, as a function of the difference in intensity rather than position:

$$I'_p = \frac{1}{W} \sum_{q \in \{ r \mid \exists I_r \}} G_{\sigma_S} (| I_p - I_q |) G_{\sigma_P} (\| p - q \|) I_q$$

(3.12)

Here $W$ is a normalisation parameter, equal to $\sum_{q \in \{ r \mid \exists I_r \}} G_{\sigma_S} (| I_p - I_q |) G_{\sigma_P} (\| p - q \|)$. Adding a weight that decreases with an increase of intensity introduces the concept of only blurring pixels of similar intensity, which effectively preserves edges due to their high
contrast. Replacing $I_p - I_q$ with pixels from another image $J_p - J_q$ gives a cross-bilateral filter, where $I$ is blurred respecting $J$'s edges.

The median-centred cross-bilateral filter is an extension of this concept. Each pixel $p$ in the new background estimate $M'$ is defined by as a weighted average of the all the pixels across the $n$ input images $I_i$, weighted by their spatial distance to $p$, and by their colour distance to the median $M$.

$$M'_p = \frac{1}{W} \sum_{i=0}^{n} \sum_{q \in \{ r | \exists I_i, r \}} G_{\sigma_S} (\|I_{i,q} - M_p\|) G_{\sigma_P} (\|p - q\|) I_{i,q}$$  \hspace{1cm} (3.13)

Intuitively, this is a smoothing of the noise in the median, by replacing each median pixel with an average of the corresponding image pixels, giving little weight to outliers. This has two positive effects. Firstly, it reduces noise in the median, giving a better background estimate. Secondly, averaging the image pixels instead of picking a pixel from one image will decrease the image favouring which causes the bitmask noise described above.

**Mask Post-Processing**

In mask post-processing, each mask is processed using two iterations of a dilation filter, where each pixel is set to the maximum value of its neighbours, followed by a Gaussian blur of radius two pixels. These values were chosen to ensure that the Gaussian blur does not interfere with the edges of the original selected areas, but instead extends them smoothly, and decreases the noise inside the selection. Extending the mask outwards fixes the second problem described above, and the Gaussian blur helps with the first problem, as it is more likely with smoother masks that a contiguous background area will come from one image.

![Figure 3.15: The result after post-processing the mask used in figure 3.14. Notice that the white outline is now gone.](image)

Most of the planned user interaction is between this stage and the next, as the mask, whose values we are setting here, is the same as the user modifiable mask described in section 3.2.
Figure 3.16: A mask before and after post-processing. The top image is the mask before dilating and blurring, whereas the bottom image is the mask after performing these operations.
3.5 Selective Blending

The selective blending stage is, as in panorama stitching, logically a weighted average. It adds to the blending in panorama stitching by having, for each image input, an associated mask of per-pixel weights. This section describes the two approaches considered for blending: linear blending and multi-band blending.

3.5.1 Linear Blending

![Figure 3.17: A linear blending of two images with masks not set to 100%. Notice the large amount of ‘ghosts’.

Linear blending is the traditional, simple form of blending used in panorama stitching. In
its basic form, linear blending sets each pixel of the new image as a weighted average of the corresponding pixels in the original images. For panorama stitching, the weights decrease linearly from the centre of the image to the edges (normalised to $[0, 1]$). This is so that there is a guaranteed smooth, linear transition between overlapping images, regardless of their relative position and orientation. This can be extended to selective blending by multiplying the pixel weights by the mask weights; then, when masked foreground overlaps with unselected background, the latter will be preferred.

This is a good method in principle, when the value of the mask over an occluding element is guaranteed to be 100%. However, if it is not, then we will get ‘ghosts’ in the final image, where the small but significant weight of the occluding element results in making the occlusion semi-transparent, instead of removing it completely (figure 3.17). This also creates blurring, ghosting effects or visible image edges for several unmodelled image parameters, such as vignetting, small misalignments, radial distortion, parallax effects, etc.

### 3.5.2 Multi-Band Blending

In addition to linear blending, this project implements multi-band blending from Burt and Adelson (1983a), as modified for more than two images in Brown and Lowe (2003), and then by me to add masking for selective blending.

**Blending Two Images**

The premise behind multi-band blending is that linearly blending two images with a mask causes a visible edge due to colour and intensity differences when blending with too hard an edge (but preserving detail near the edges), and visible ghosts of detail when blending with too soft an edge (but creating a smooth transition in colour and intensity). Burt and Adelson (1983a) instead proposed to split the image into bands in its frequency domain, and blend different bands with different edge softness. Then, there would be a hard edge for high-frequency details, eliminating ghosts, but properties with low frequency, such as average colour and intensity, will be blended into each other for a smooth transition.

Splitting an image into bands is performed by band-pass filtering the image using a Difference of Gaussians. The image is successively low-pass filtered by a Gaussian blur of increasingly large radius (figure 3.18). This collection of blurred images is called a Gaussian pyramid, for reasons described shortly. We then calculate the bands as the difference between successive low-passed images—this approximates a band-pass with a Laplacian of Gaussian function, and is therefore called a Laplacian pyramid (figure 3.19). Then, the lower bands are blended over a larger area than the higher bands, where the

---

7Where image intensity decreases towards the edges of an image
highest frequency band is blended with a bitmask. The final image is obtained by simply summing the bands.

\[ G_0 = I \]  
\[ G_i = g_{\sigma_i} \ast G_{i-1} \quad (0 < i \leq n) \]  
\[ L_i = G_i - G_{i+1} \quad (0 \leq i < n) \]  
\[ L_n = G_n \]  

This can be expressed mathematically as:

Figure 3.18: The first six low-pass filtered images. The first level is the original image.

where \( I \) is the input image, \( n \) is the number of levels of the pyramid, \( \ast \) represents convolution, and \( g_\sigma \) is the Gaussian function with variance \( \sigma^2 \). Also, it can be shown that the
Figure 3.19: Converting a Gaussian pyramid to a Laplacian pyramid by subtracting layers. The lowest level of the Laplacian pyramid is a low-pass filter, and is identical to the lowest level of the Gaussian pyramid.
The sum of the Laplacians gives us the original image:

\[
\sum_{i=0}^{n} L_i = L_0 + L_1 + \cdots + L_n \\
= (G_0 - G_1) + (G_1 - G_2) + \cdots + (G_{n-1} - G_n) + G_n \\
= G_0 = I
\]  

(3.18)

Burt and Adelson (1983a) suggested that when blending two images by masking, that one can create a black and white (with no grey) mask for the high frequency detail, and then create a Gaussian pyramid for this mask that gives the softer masks for the lower bands. The intensity of the mask at a pixel would give the two weights used for linearly blending the appropriate pixels of the two images.

**Blending More Than Two Images**

Brown and Lowe (2003) extended this concept to blending an arbitrary number of images. Once the images’ pixels are weighted by their distance from the centre of the image, we create a bitmask for each image where the image with maximum weight at a pixel will have a 1 in the bitmask, and all other images will have a 0 in their mask for that pixel. The rest of the algorithm proceeds as in Burt and Adelson (1983a), except that each image has its own Gaussian pyramid of masks which it provides for the linear blending.

In my project, I extend this idea by creating bitmasks not only based on distances to the centre, but also based on the values of the masks of weights at that point. This has the significant advantage over linear blending that we only care about the maximum weight; therefore, only relative mask values matter, not absolute values, which means we can avoid having to normalise at the anomaly detection stage.

**Image Pyramids**

The implementation of this is very memory-efficient; it is an image pyramid (Burt and Adelson, 1983b). Instead of keeping a copy of each blurred image, and convolving with ever larger Gaussian kernels, we notice that the low-pass filtering removes information from the image; therefore, we should be able to down-sample the image, and recursively apply the same size kernel to ever smaller images. Since the images are smaller at each level of detail, this is called a pyramid.

The fixed size kernel means that it can be precalculated and stored. This kernel can be designed to exactly halve the dimensions of the image at each application, while approximating a Gaussian blur (figure 3.20). Halving the dimensions allows the pyramid construction code to be very simple, by essentially ‘skipping’ every other row and column of pixels at each layer of the pyramid (figure 3.21).
Figure 3.20: The first six levels of a Gaussian pyramid, identical in information content to those in figure 3.20, but resampled due to loss of information content in lower levels.

Figure 3.21: A one-dimensional graphical representation of the construction of Gaussian pyramid layers from the layer below.
The Laplacian is therefore also constructed as a pyramid, by enlarging (and interpolating) a lower level to perform the subtraction. In this project, to conserve the surprisingly limited memory when performing this operation in parallel, the Laplacian pyramid is constructed from the Gaussian pyramid destructively; in other words, the Gaussian pyramid’s images are traversed downwards, and replaced by the Laplacian when they are no longer needed.

### 3.6 Summary

In this chapter, I have described the implementation of my project. I have described the core classes creating a framework, in particular linear algebra and bitmap manipulation. I have also described the design and build of the graphical user interface, including the functionality where a user may manually draw masks on images.

Furthermore, I have talked about the image processing pipeline, split into three stages: aligning the images, detecting and selecting occluding foreground elements, and blending the images whilst removing the selected areas. Images are aligned by finding feature points, finding matches between images to find homographies between images, and then aligning based on those homographies. Occluding elements are detected by estimating the background, and selecting deviations from this estimate. The images are then selectively blended with multi-band blending, which blends different frequency bands in the image by different amounts to keep detail sharp, but allow smooth transitions in intensity and colour.
Evaluation

This chapter deals with the evaluation of the project. Since the alignment is independent of the actual background extraction, I have decided to evaluate it separately. I also provide qualitative examples of the project working, and have performed a user study comparing several methods presented in this paper, namely comparing manual and automatic occlusion detection, linear and multi-band blending, and simply taking the median.

4.1 Framework

White-box testing, where the application’s internals are known when selecting test data, allowed me to rigorously test the matrix and vector classes during their writing. These results were compared to results in MATLAB which were assumed to be correct; this allowed me to test for the matrix and vector code with known expected output, and white-box testing allowed me to select the input data to be minimal examples, to simplify testing. I performed similar tests with the matrix decomposition classes.

I also implemented black-box unit tests for the SVD, which checked if the resulting decomposition matrices have the expected properties. Several random matrices are created, and have their SVD calculated. The matrices are then checked (as appropriate) if they are unitary, diagonal, and if their product is, within some small tolerance, the original matrix.

This is a ‘monte-carlo’ method of testing, as it is impossible to test all the possible input matrices. Instead, I test a random sample of input matrices. If their results are correct, then I assume that that there is a high probability that the remaining matrices are correct. White-box testing, as described above, allows us to find special cases separately.

4.2 Alignment

I decided to evaluate the alignment quantitatively using known matrix distortions. I applied several known matrices to an image (figure 4.1) to create a set of distorted images. The matrices were selected as several affine transforms, for which we expect the feature detection to work well as it is theoretically invariant under affine transforms. Also, a ‘difficult’, projective transform matrix was used, to test the detectors robustness to projective transforms.
These distorted images were then passed into the alignment stage, one picture at a time, alongside the original. The homography between the distorted image and the original was calculated and stored. Ideally, the homography should be the exact inverse of the distortion matrix, and their product should therefore be the identity. I measure the error of the homography as its distance to the identity, using the Frobenius norm. The Frobenius norm is a measure of a matrix, similar to a vector's magnitude, defined as:

$$\|M\|_F = \sqrt{\sum_{i,j} |M_{ij}|^2}$$

(4.1)

I define the distance between two matrices as the Frobenius norm of their difference.

### 4.2.1 Results

Table 4.1 summarises the alignment results after applying several different distortion matrices to figure 4.1. The full results are in appendix B.

It appears that the greatest error is in the projective transformations. This is to be expected, as the SIFT algorithm is not invariant under projective transformations, therefore some degree of accuracy is lost. Of the affine transforms, skewing has the largest error, and scaling the second largest; this is fortunate, as the transformations we expect from minor movements of the camera approximate rotation and translation. Regardless of the ordering, all the errors are within two orders of magnitude of each other, therefore this is an excellent result confirming the robustness of the image alignment.
Table 4.1: The results of the alignment evaluation. Matrix values have been rounded for legibility.
4.3 Examples

This section consists of several examples of the implementation’s results when applied to real life photos. All the photographs were taken using a handheld digital SLR camera, without a tripod, but with constant aperture, shutter speed, white balance and focus. This ensured that the camera parameters were consistent with the assumptions of this project.

Figure 4.2: The Cambridge Bookshop. The automatic anomaly detection was mostly successful, however small manual adjustments were necessary to remove a floating torso where there had not been a majority of background in the set of photographs.
Figure 4.3: King's College, Cambridge. In this case, the anomaly detection was fully automatic.
Figure 4.4: The Cambridge Guildhall. The anomaly detection was partially automatic, however manual adjustments were necessary to remove a cyclist who was unlocking his bike in the majority of photographs.
4.4 User Study

The nature of this project is such that its overall success is difficult to measure objectively. The ultimate aim of the project is to create appealing images, where being appealing is a highly subjective property that is effectively impossible to define mathematically. Therefore, this section details a user study that was carried out to obtain subjective, third-party opinions, to evaluate the success of the project.

The original project plan did not allow time for a user study. However, as the project reached its concluding stages, an analysis of the evaluation criteria showed that a user study would be essential for a fair and complete evaluation.

4.4.1 Preparation

The user study was based around three sets of six printed photographs. Each set represented one location (figures 4.2, 4.3 and 4.4), where several photographs had been taken to attempt to reconstruct the background. One of the photographs was a copy of one of the originals, picked at random. The originals were automatically aligned, and the other five photographs in the set were generated by the implementation of this project, using several of the methods presented above. These methods, along with abbreviations I will use for clarity, were:

- A simple median (Med).
- Manually selected occlusions, followed by linear blending (ML).
- Manually selected occlusions, followed by multi-band blending (MB).
- Automatically selected occlusions, followed by linear blending (AL).
- Automatically selected occlusions, followed by multi-band blending (AB).

These were printed at normal photograph printing size (6 inches by 4 inches) on photo paper, using a home inkjet printer. This was assumed to be a good estimate of how an implementation of this project would be used in real life. To identify the method used for each photograph in the set, the photographs were assigned a letter from A–F. The letter for each method was picked at random for each set, to decrease bias.

The study participants were then asked to order the six photographs in each set from best to worst, according to three different criteria: appeal, quality and number of artifacts. The full questionnaire is available in appendix C.

4.4.2 Results

The user study was performed by 12 people over a 5 hour session. Informal comments during the study suggested that the variance between processed images was quite low,
however certain trends became apparent in the analysis of the results. Also, it quickly became obvious that including one of the original images in the ranking was troublesome for people, in particular for the latter two orderings, as it was not obvious if occlusions (in the form of people) should be classed as artifacts. Therefore, the original images’ ranks have been ignored for this analysis.

The resulting rankings are analysed in two ways. Firstly, the frequency of each ranking is plotted, for each method (figure 4.5). This gives a general overview of the ranking tendencies. For example, it is immediately obvious that AL, automatically selected occlusions followed by linear blending, are considerably worse than the remaining methods. This is most likely due to the ghosting effect seen in figure 3.17. Also, the manual selection methods, ML and MB, are visibly more preferred than the automatic selection methods. This is to be expected, as the manual selection can be considered the ‘perfect’ selection, which the automatic selection attempts to reach. Interestingly, the linear and multi-band blending methods seem to be approximately equally good when manual blending is performed, though the multi-band blending was ranked first more often.

The second way this is analysed is by creating a matrix of pairwise comparisons – that is, for $M_{x,y}$, the value is how many times $x$ was ranked better than $y$. Table 4.2 shows the comparison matrices created from these data. Each cell specifies how many times the method from its row was ranked as better than the method from its column. The ‘Total’ column describes how many times the row method was ranked higher than any other method. The matrix confirms that MB was the best method, ‘defeating’ ML approximately 60% of the time. The matrix also shows that Med and AB are ranked very
similarly, with neither defeating the other with any significant regularity. It is also obvious that AL is the worst method.

Table 4.2: Pairwise comparison matrices for the relative rankings of background extraction methods in each question, as well as a total.

The conclusions we can draw from this user study are that manual detection is still better than automatic detection (table 4.3); however, the participants comments suggest that, although visible, the differences are small. Furthermore, we see that multi-band blending is superior to linear blending in both manual and automatic detection (table 4.3), though especially in the latter. The simple median seems to be as effective as automatic detection followed by multi-band blending—which is to be expected, given that the automatic...
### Table 4.3: Pairwise comparison matrices for automatic vs manual selection, and linear vs multi-band blending.

<table>
<thead>
<tr>
<th></th>
<th>Automatic vs Manual</th>
<th>Linear vs Multi-Band</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ML+MB</td>
<td>AL+AB</td>
</tr>
<tr>
<td>ML+MB</td>
<td></td>
<td>73</td>
</tr>
<tr>
<td>AL+AB</td>
<td>359</td>
<td></td>
</tr>
</tbody>
</table>

Detection relies on the median. However, one must also remember that the automatic detection can be manually amended, whereas the simple median can not. Therefore, automatic detection with multi-band blending is as good as the median by default, and superior to it in general thanks to its flexibility.
5

Conclusions

5.1 Results

The project has been an overall success. All the core project aims have been completed (save for manual alignment, which was deemed unnecessary), and the panorama stitching extension has been implemented. The project performs its purpose, automatic background extraction from a set of photographs, with excellent results. It successfully adapts two dissimilar technologies, panorama stitching and motion detection, to create a novel way of extracting backgrounds that is also flexible in allowing manual input.

The project presents a simple graphical user interface, where a photographer can easily load several images of a location, and step through the stages of alignment, anomaly detection and blending, with little effort. A photographer can also amend the automatic occlusion detection, for example to manually select difficult regions (such as crowds), or to unmask a previously selected occlusion. This is useful if the photographer wants certain occluding elements in the photograph, for example friends and family.

5.2 Lessons Learnt

In retrospect, although the research and preparation done was sufficient for a successful project, more research into related work could have affected my choice of algorithms. Also, the panorama stitching extension was implemented later than originally planned, due to unexpected problems arising in the interaction between anomaly selection and blending.

With the benefit of hindsight, I would dig deeper into current research, in particular in the field of motion detection, whose more recent advances in background estimation could have been used in this project also. I would also prepare the user study earlier, to allow for multiple pilot studies and a full statistical analysis of the results.

5.3 Future Work

There are several ways in which this project could be continued. Recent work on into background estimation could provide better anomaly detection, and newer research into
feature detection (such as Bay et al., 2006) and nearest-neighbour searches (Muja and Lowe, 2009) could significantly speed up the alignment stage.

The anomaly detection could be enhanced using such techniques as blob detection to select large contiguous anomalous areas, and discard smaller ones as noise. Using statistical methods, one could also select the background in areas where it is not in the majority of photographs, by comparing a pixel with its surroundings (using a heuristic that backgrounds tend to be contiguous), or by using a clustering analogy to the mode, rather than the point centre analogy to the median.

In areas where the background cannot be extracted—for example a parked car, a crowd of people, or people sitting down—in-painting could be used to fill in the gaps. It would be an interesting project to try to integrate in-painting with the logic of the pipeline as described in chapter 3.

Also, further work could be done on adapting the algorithms used in this project to parallelisable instructions that can be run on the GPU, as its processing power is currently underused.

There could also be work on adding currently unmodelled parameters into this project. One could correct for different white-balance between photographs, or different intensities caused by changes in shutter speed or aperture. Furthermore, one could model camera translation to allow for parallax effects in the photographs. Conceivably, one could take this as far as extracting different depth layers instead of just the background.


Appendices

A SIFT

The Scale-Invariant Feature Transform (SIFT) uses a Difference of Gaussians (DoG) approach to locate local scale-space extrema, which are stable features across all possible scales. The Difference of Gaussians approach is, briefly, a computationally efficient approximation of the Laplacian of Gaussian function, by repeated blurring of an image (DoG is used again, and explain in more detail, in section 3.5.2). Experimentally, the Laplacian of Gaussian has proven to give stable image features (Mikolajczyk, 2002).

Using DoG, SIFT locates local extrema in the scale space, which are the maximum or minimum of their eight neighbours in the current scale, and of the nine neighbours in the each of the adjacent scales. It then performs a detailed fit to the nearby data for location, scale, and ratio of principal curvatures. This information allows points to be rejected that have low contrast (and are therefore sensitive to noise) or are poorly localized along an edge.

Figure A.1: Generating a $2 \times 2$ feature descriptor (where each cell is actually 8 values) from an $8 \times 8$ set of samples. The gradient magnitude and orientation at each image sample point in a region around the keypoint location is computed. These are weighted by a Gaussian window, represented by the overlaid circle. Each $4 \times 4$ subregion of samples is accumulated into an 8-valued orientation histogram, which becomes a cell in the feature descriptor. Image taken from Lowe (2004).

Finally, SIFT assigns each feature point an orientation, based on local image gradient directions, and creates a 128-value feature descriptor that is as invariant to change in illumination or 3D viewpoint as possible using the local image gradients (figure A.1).
## B Full Alignment Results

This section lists the full results of the image alignment evaluation from section 4.2. In each case, the distortion matrix is the known matrix by which the original input image is transformed. The calculated homography is the homography that the implementation calculates given the distorted image and the original image. This should theoretically be the inverse of the distortion matrix, therefore their product should be the identity. The error is measured as the distance to the identity, by finding the Frobenius norm of their difference.

### B.1 Translation

| Distortion Matrix $m$ | \[
\begin{pmatrix}
1 & 0 & 100 \\
0 & 1 & 200 \\
0 & 0 & 1
\end{pmatrix}
\] |
|---|---|
| Calculated Homography $h$ | \[
\begin{pmatrix}
9.99 \times 10^{-1} & 5.24 \times 10^{-4} & 3.77 \times 10^{-2} \\
2.22 \times 10^{-4} & 1.00 & 1.05 \times 10^{-2} \\
-2.24 \times 10^{-6} & 6.13 \times 10^{-6} & 1.00
\end{pmatrix}
\] |
| Product $h \cdot m$ | \[
\begin{pmatrix}
9.99 \times 10^{-1} & 5.24 \times 10^{-4} & 3.77 \times 10^{-2} \\
2.22 \times 10^{-4} & 1.00 & 1.05 \times 10^{-2} \\
-2.24 \times 10^{-6} & 6.13 \times 10^{-6} & 1.00
\end{pmatrix}
\] |
| Difference to Identity $h \cdot m - I$ | \[
\begin{pmatrix}
-8.36 \times 10^{-4} & 5.24 \times 10^{-4} & 3.77 \times 10^{-2} \\
2.22 \times 10^{-4} & 2.58 \times 10^{-4} & 1.05 \times 10^{-2} \\
-2.24 \times 10^{-6} & 6.13 \times 10^{-6} & 0.00
\end{pmatrix}
\] |
| Error $\|h \cdot m - I\|_F$ | 0.04 |
### B.2 Scaling

<table>
<thead>
<tr>
<th>Distortion Matrix</th>
<th>$m$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\begin{pmatrix} 0.6 &amp; 0 &amp; 0 \ 0 &amp; 0.6 &amp; 0 \ 0 &amp; 0 &amp; 1 \end{pmatrix}$</td>
</tr>
<tr>
<td>Calculated Homography</td>
<td>$h$</td>
</tr>
<tr>
<td></td>
<td>$\begin{pmatrix} 1.66 &amp; -4.19 \times 10^{-4} &amp; 3.68 \times 10^{-1} \ -3.81 \times 10^{-3} &amp; 1.66 &amp; 9.30 \times 10^{-2} \ -2.40 \times 10^{-5} &amp; -6.46 \times 10^{-6} &amp; 1.00 \end{pmatrix}$</td>
</tr>
<tr>
<td>Product</td>
<td>$h \cdot m$</td>
</tr>
<tr>
<td></td>
<td>$\begin{pmatrix} 1.01 &amp; 2.91 \times 10^{-1} &amp; -4.16 \times 10^{-1} \ 9.99 \times 10^{-2} &amp; 1.60 \times 10^{-1} &amp; 6.01 \times 10^{-2} \ 9.67 \times 10^{-6} &amp; -1.50 \times 10^{-5} &amp; 1.00 \end{pmatrix}$</td>
</tr>
<tr>
<td>Difference to Identity</td>
<td>$h \cdot m - I$</td>
</tr>
<tr>
<td></td>
<td>$\begin{pmatrix} 1.00 \times 10^{-1} &amp; 9.99 \times 10^{-1} &amp; 6.01 \times 10^{-2} \ 9.67 \times 10^{-6} &amp; -1.50 \times 10^{-5} &amp; 1.00 \end{pmatrix}$</td>
</tr>
<tr>
<td>Error</td>
<td>$| h \cdot m - I |_F$</td>
</tr>
<tr>
<td></td>
<td>0.38</td>
</tr>
</tbody>
</table>

### B.3 Skewing

<table>
<thead>
<tr>
<th>Distortion Matrix</th>
<th>$m$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\begin{pmatrix} 1 &amp; 0.3 &amp; 0 \ 0.1 &amp; 1 &amp; 0 \ 0 &amp; 0 &amp; 1 \end{pmatrix}$</td>
</tr>
<tr>
<td>Calculated Homography</td>
<td>$h$</td>
</tr>
<tr>
<td></td>
<td>$\begin{pmatrix} 1.01 &amp; 2.91 \times 10^{-1} &amp; -4.16 \times 10^{-1} \ 1.00 \times 10^{-1} &amp; 9.99 \times 10^{-1} &amp; 6.01 \times 10^{-2} \ 9.67 \times 10^{-6} &amp; -1.50 \times 10^{-5} &amp; 1.00 \end{pmatrix}$</td>
</tr>
<tr>
<td>Product</td>
<td>$h \cdot m$</td>
</tr>
<tr>
<td></td>
<td>$\begin{pmatrix} 1.00 \times 10^{-1} &amp; 9.99 \times 10^{-1} &amp; 6.01 \times 10^{-2} \ 9.67 \times 10^{-6} &amp; -1.50 \times 10^{-5} &amp; 1.00 \end{pmatrix}$</td>
</tr>
<tr>
<td>Difference to Identity</td>
<td>$h \cdot m - I$</td>
</tr>
<tr>
<td></td>
<td>$\begin{pmatrix} 1.00 \times 10^{-1} &amp; -6.84 \times 10^{-4} &amp; 6.01 \times 10^{-2} \ 9.67 \times 10^{-6} &amp; -1.50 \times 10^{-5} &amp; 0.00 \end{pmatrix}$</td>
</tr>
<tr>
<td>Error</td>
<td>$| h \cdot m - I |_F$</td>
</tr>
<tr>
<td></td>
<td>0.52</td>
</tr>
</tbody>
</table>
B.4 Rotation

<table>
<thead>
<tr>
<th>Distortion Matrix</th>
<th>$m$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\begin{bmatrix} 0.87 &amp; -0.5 &amp; 20 \ 0 &amp; 0.87 &amp; 30 \ 0 &amp; 0 &amp; 1 \end{bmatrix}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Calculated Homography</th>
<th>$h$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\begin{bmatrix} 8.65 \times 10^{-1} &amp; 4.97 \times 10^{-1} &amp; -9.28 \times 10^{-3} \ -4.96 \times 10^{-1} &amp; 8.65 \times 10^{-1} &amp; -3.28 \times 10^{-2} \ 2.44 \times 10^{-5} &amp; 2.19 \times 10^{-6} &amp; 1.00 \end{bmatrix}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product</th>
<th>$h \cdot m$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\begin{bmatrix} 7.75 \times 10^{-4} &amp; 1.00 &amp; -3.28 \times 10^{-2} \ 2.23 \times 10^{-5} &amp; -1.03 \times 10^{-5} &amp; 1.00 \end{bmatrix}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Difference to Identity</th>
<th>$h \cdot m - I$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\begin{bmatrix} 1.01 \times 10^{-3} &amp; -1.47 \times 10^{-4} &amp; -9.28 \times 10^{-3} \ 7.75 \times 10^{-4} &amp; 6.51 \times 10^{-4} &amp; -3.28 \times 10^{-2} \ 2.23 \times 10^{-5} &amp; -1.03 \times 10^{-5} &amp; 0.00 \end{bmatrix}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Error</th>
<th>$|h \cdot m - I|_F$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$0.03$</td>
</tr>
</tbody>
</table>

B.5 Projection

<table>
<thead>
<tr>
<th>Distortion Matrix</th>
<th>$m$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\begin{bmatrix} 1.2 &amp; 0.1 &amp; 20 \ 0.05 &amp; 1.2 &amp; 30 \ 5 \times 10^{-4} &amp; 5 \times 10^{-4} &amp; 1 \end{bmatrix}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Calculated Homography</th>
<th>$h$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\begin{bmatrix} 8.23 \times 10^{-1} &amp; -6.65 \times 10^{-2} &amp; -1.44 \times 10^{1} \ -2.51 \times 10^{-2} &amp; 8.15 \times 10^{-1} &amp; -2.25 \times 10^{1} \ -4.09 \times 10^{-4} &amp; -3.95 \times 10^{-4} &amp; 1.00 \end{bmatrix}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product</th>
<th>$h \cdot m$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\begin{bmatrix} -6.28 \times 10^{-4} &amp; 9.84 \times 10^{-1} &amp; 1.45 \ -1.06 \times 10^{-5} &amp; -1.54 \times 10^{-5} &amp; 1.00 \end{bmatrix}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Difference to Identity</th>
<th>$h \cdot m - I$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\begin{bmatrix} -3.46 \times 10^{-3} &amp; -4.88 \times 10^{-3} &amp; 6.84 \times 10^{-2} \ -6.28 \times 10^{-4} &amp; 1.59 \times 10^{-2} &amp; 1.45 \ -1.06 \times 10^{-5} &amp; -1.54 \times 10^{-5} &amp; 0.00 \end{bmatrix}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Error</th>
<th>$|h \cdot m - I|_F$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$1.45$</td>
</tr>
</tbody>
</table>

C User Study Details

Participants of the user study were presented with a user study form (figure C.2) and the three sets of photographs, each photograph labeled with its letter A–F, and the number of its set. The form includes an informed consent form to sign before participating; this ensures that there are no ethical issues. The questions presented were derived from a pilot
study with two people, with two similar questions but on a different set of photographs. Their comments and my observations resulted in the final questionnaire.

Automatic People Removal from Photographs

User Study

Thank you for participating in this user study. Please sign the below consent form before continuing.

I wish to participate in a user evaluation study which is being conducted by Lech Świrski, a Computer Science student at the University of Cambridge Laboratory. The purpose of this research is to assess the results of work carried out during a final year project.

The study involves subjective judgments about provided images. My name will not be identified at any time, but comments I make may be anonymously quoted. I understand that I am free to withdraw from participation at any time without penalty.

Signed

You have been given three sets of six photographs, from three different locations, each labeled on the back with a letter A–F. Five of these are artificial photographs, created from a set of originals using five different methods; the sixth is one of the original photos.

For each set of six photographs, please consider their order, from best to worst, for the following three questions:

1. Which photograph is most appealing, so that you would be likely to keep it, for example, in your photo album?

2. Which photograph has the highest image quality, for example image sharpness and lighting?

3. Which photograph has the fewest ‘artefacts’, that is the fewest mistakes such as ‘ghosts’, partially removed people and random pixels?

Please record your orderings, using the letters on the backs of the photographs, in the tables below:

<table>
<thead>
<tr>
<th>Set 1. King’s College, Cambridge</th>
<th>Appeal</th>
<th>Quality</th>
<th>Artefacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appeal</td>
<td>1.</td>
<td>1.</td>
<td>1.</td>
</tr>
<tr>
<td>2.</td>
<td>2.</td>
<td>2.</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>3.</td>
<td>3.</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>4.</td>
<td>4.</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>5.</td>
<td>5.</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>6.</td>
<td>6.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set 2. Cambridge Guildhall</th>
<th>Appeal</th>
<th>Quality</th>
<th>Artefacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appeal</td>
<td>1.</td>
<td>1.</td>
<td>1.</td>
</tr>
<tr>
<td>2.</td>
<td>2.</td>
<td>2.</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>3.</td>
<td>3.</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>4.</td>
<td>4.</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>5.</td>
<td>5.</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>6.</td>
<td>6.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set 3. Cambridge Bookshop</th>
<th>Appeal</th>
<th>Quality</th>
<th>Artefacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appeal</td>
<td>1.</td>
<td>1.</td>
<td>1.</td>
</tr>
<tr>
<td>2.</td>
<td>2.</td>
<td>2.</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>3.</td>
<td>3.</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>4.</td>
<td>4.</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>5.</td>
<td>5.</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>6.</td>
<td>6.</td>
<td></td>
</tr>
</tbody>
</table>

Figure C.2: The user study form presented to participants.
Computer Science Project Proposal

Removing People from Photographs

22 October 2008

Project Originator: This was a project suggestion by Christian Richardt

Resources Required: See attached Project Resource Form

Project Supervisor: Christian Richardt

Signature:

Director of Studies: Pietro Lió

Signature:

Overseers: Frank Stajano and David Greaves

Signatures:
Introduction and Description of the Work

A common problem when taking photographs of public locations is that the view of at least one part of the targeted object is invariably blocked by a pedestrian. However, under the assumption that pedestrians move whereas stationary objects do not, with sufficient photos of the target one should have enough data to construct an unobstructed image of the target.

If the photographer is suitably prepared, then he might bring a tripod and use the same settings\(^1\) on his camera to make a manual construction of the photo near trivial, by using simple masking techniques. Problems arise when the photographer is not so well prepared: if the photos are from slightly different vantage points, taken with different settings (which is indeed likely for most cameras when set to *auto*), or indeed if the pedestrian selection is to be automatic.

This project aims to resolve these problems in (as far as possible) an automated manner. It would have to align the images as necessary, select the pedestrians and then replace them with the corresponding area from another photo, all in such a way as for the resulting image to seem to be authentic.

Resources Required

No particular special resources are required for this project, though I am allowing for third-party libraries if they are suitable. Their use will be documented accordingly.

Starting Point

I have already had some experience with object recognition for the Cambridge Autonomous Underwater Vehicle project, where I wrote line and circle Hough transforms, a rectangle (‘validation gate’) finder based on the line Hough transform, and some fairly simple colour-filter based object detection. I also started a Difference-of-Gaussians blob detection algorithm, but this was not finished due to time constraints.

This was all written in Java, a language I had not used before University but have used extensively since. I also have a large amount of experience with C\#\(^2\), mostly in web development in ASP.NET and more recently with Silverlight.

In addition to this programming experience, many of the tasks I wish to automate I have also previously done manually, hence I can appreciate the unexpected problems that can arise.

---

\(^1\)Aperture, exposure time, white balance etc.

\(^2\)Programmer, development, etc.
Substance and Structure of the Project

This project would involve automating several tasks which are surprisingly natural to humans. These are image alignment, anomalous object selection and image blending. Feature detection in images is a well researched subject, in particular the SIFT algorithm; using simple affine and perspective transforms and assuming the problem presented by parallax is negligible, it should be possible to align two or more images automatically.

Matching exposure and white balance in a set of images should also be possible, either by using meta data in the image itself or by attempting to normalize them by ‘eye’, e.g. using histogram equalization or some iterative algorithm based on the standard deviations between the images’ pixel values. Selecting pedestrians (and other anomalous objects such as pigeons) is plausible, possibly by looking at outliers in the distribution of a pixel’s values across the set of image, and perhaps augmented by some form of blob detection.

Once the anomalies are selected, they should be replaced with data from another image (or several other images) in such a way as to be seamless without closer inspection. The colour matching performed earlier will be helpful, however this will not be enough and will require some additional blending.

Any and all of the above methods are imperfect, and could give very unexpected and very wrong results; hence, each stage of the process should have a human in the loop to make manual changes. In particular, the alignment can go spectacularly wrong, as can anomaly selection — one should be able to manually transform the images, and to ‘paint’ wanted or unwanted areas.

Possible extensions are in-painting any regions which are identified as pedestrians but where alternatives are not available, and extending the power of the image alignment and blending into a panorama stitcher.

Success Criterion

To demonstrate the successful automatic removal of pedestrians from a suitably ‘easy’ set of photos (where the object in question is almost wholly visible in most or even all photos), as well as a semi-automatic removal of pedestrians from a ‘harder’ set of photographs where some human aid may be necessary.

Timetable and Milestones

Dividing the weeks after the proposal is submitted into 10 sets of 3 weeks:
**Weeks 1 to 3 (ending 7 Nov 2008)**

Reading about different methods of feature detection, for both alignment and anomaly detection. Comparing the image handling facilities of Java and C# to select which to use, and writing code to display multiple images in whichever is selected. Start writing code to ‘paint’ over sections of the image.

Milestones: A working image displaying GUI written in the language of choice.

**Weeks 4 to 6 (ending 28 Nov 2008)**

Creating a GUI to manually select anomalies in an image, and writing code to blend in a section of another image into the selected region, assuming the images are already aligned and colour-matched. Also start writing code for manual image alignment.

Milestones: A GUI which allows one to feed in aligned images, and can blend segments from one image to another based on user selection.

**Weeks 7 to 9 (ending 19 Dec 2008)**

Write code which detects pedestrians in aligned, colour-matched images, and selects the areas in each photo that need to be removed in a way that allows the user to modify this selection using the previously created GUI. Start writing code for feature detection in an image.

Milestones: Having an anomaly detection algorithm which can be used as the penultimate step of the process.

**Weeks 10 to 12 (ending 9 Jan 2009)**

Write code that aligns a set of images, using feature detection and affine/perspective transforms. If the feature detection is not working near the end of this section, find a library to put there as a (possibly permanent) placeholder. Integrate this with the manual alignment.

Milestones: Image alignment working.

**Weeks 13 to 15 (ending 30 Jan 2009)**

Depending on the state of the project so far, either improve the main project code so that the effect is presentable or begin work on panorama stitching.
Milestones: A fairly well working pedestrian removal tool. Progress Report submitted and entire project reviewed both personally and with Overseers.

**Weeks 16 to 18 (ending 20 Feb 2009)**

Depending on previous progress, either work on the main code, the panorama stitching, or begin work on in-painting.

Milestones: Code that would be acceptable for submission save for minor cosmetic changes, possibly one or both of the proposed extensions.

**Weeks 19 to 21 (ending 13 Mar 2009)**

Begin writing the Dissertation, try to make only minor changes to the functionality.

**Weeks 22 to 24 (ending 3 Apr 2009)**

Submit the Dissertation draft, lock the functionality of the code and only work on internal improvements (such as efficiency, accuracy and bugs).

**Weeks 25 to 27 (ending 24 Apr 2009)**

Finish the Dissertation as much as possible. Review the whole project, polish out as many remaining bugs as possible.

**Weeks 28 to 30 (ending 15 May 2009)**

Aim to submit the Dissertation near the start of this period.

Milestone: Submission of Dissertation.