An Introduction To Visual Understanding Techniques

UNIVERSITY OF BATH

DEPARTMENT OF COMPUTER SCIENCE: CS50244
CANDIDATE NUMBER: 02151
18/01/2018

1 Introduction

This report aims to discuss and implement methods over six fundamental topics within visual understanding. Each topic will be broken into three sections, a review of each exercise, the implementation, and experimentation. The topics are covered in the following order:

1. Convolution
2. Features
3. Matching
4. Camera calibration
5. Fundamental matrix estimation
6. Triangulation of 3D points

Within the review section each exercise will have its methods discussed as well as a brief introduction to the theory and algorithms used. A brief review of the background literature will be included. The implementation will then be discussed using code snippets and pseudocode to explain the approach taken to the implementation.

The experimentation section will give an account of the experiments conducted, as well as any hypothesis that are tested. Conclusions will be drawn as well as a discussion on potential improvements and alternative approaches to the implementation.

2 Convolution

2.1 Review of Convolution

Convolution is the process of adding each pixel of the image to its local neighbors by some weighted value. The weighting is performed by a kernel, the kernel can take various forms depending on the intended effect. Examples of kernels can be seen in equation 1.

$$\begin{aligned} Blur &= \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \\ Sharpen &= \begin{bmatrix}0 & -1 & 0\\ -1 & 5 & -1\\ 0 & -1 & 0\end{bmatrix} \end{aligned} \tag{1}$$

The most basic form of convolution (entitled basic convolution for the remainder of this report) involves iterating through all pixels in an image and applying the kernel as a "window" over the pixel. The new pixel value takes the sum of the kernels contributions from all neighbors covered by the window.

This basic convolution does not take into account the borders of an image. One method of dealing with this is to extend the edges of the image. This implementation is entitled extended convolution and functions when the kernel overlaps an edge of the image. The nearest edge pixel is then copied and extended as far as necessary to fill the kernel window. This convolution technique can be used for various image filtering applications by using various kernels. Blur and sharpen kernel were shown in equation 1, an alternative approach to the sharpening kernel is to compute the blur convolution for the image and subtract it from the original image. More complex kernels can be used such as edge detection kernels. Those implemented in this paper are the Sobel operator kernels [1], shown in equation 2. The Sobel operator, although producing effective low cost edge identification, does struggle for high frequency variations in the image.

$$\begin{aligned} Horizontal &= \begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix} \\ Vertical &= \begin{bmatrix}1 & 0 & -1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix} \\ Diagonal &= \begin{bmatrix}-2 & -1 & 0\\ -1 & 0 & 1\\ 0 & 1 & 2\end{bmatrix} \end{aligned} \tag{2}$$

A more advanced blurring kernel is possible to by implementing Gaussian kernels. This allows the degree of blur to be controlled simply by changing the standard deviation used in the Gaussian distribution. In two dimensions, as in a 2D grayscale image, the Gaussian function is as follows:

$$G(x,y) = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3}$$

where x and y are the distances from the origin, and σ is the standard deviation parameter [2]. The

kernel is then produced by sampling the distribution across the desired kernel size; for example if a kernel size of 10 is required, $G(x, y)$ will be calculated for integer values of x and y between -5 and 5 to give a square kernel of size 10.

The convolution technique can be instead applied in the frequency domain via Fourier transforms. In this case, both the kernel and image undergo a Fourier transform from the spatial domain into the frequency domain. The transformed kernel is then applied in the frequency domain. Due to the convolution theorem, this is much faster than the basic convolution described earlier. The convolution theorem is as follows:

$$\begin{aligned} \mathcal{F}\{f * g\} &= \mathcal{F}\{f\} \cdot \mathcal{F}\{g\} \\ f * g &= \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\} \end{aligned} \tag{4}$$

where $\mathcal{F}$ indicates the Fourier transform operator, f and g represent the image and kernel. A full derivation of the convolution theorem can be found in Arfken's work [3]. As is shown in equation 4, convolution in the spatial domain is equivalent to element wise multiplication in the frequency domain. The multiplication process is computationally faster so transforming f and g and computing in frequency space is significantly faster for large images or kernels than a direct convolution.

2.1.1 Convolution Implementation

For this implementation, as well as all others in this report, a grey scale input image is used. The basic convolution function can be summarized as the following pseudocode.

Result: Returns the convolution between an image (I) and a kernel (K)
for *Pixel in I* **do**
  Center K on pixel;
  Sum = 0;
  for *Each Pixel covered by element in K* **do**
    Sum = Sum + (covered pixel * element in K);
  end
  Pixel = Sum;
end

Algorithm 1: Basic Convolution

This is then extended simply by adding an "if" statement to the center loop to check to see if the covered pixel is outside of the image, if it is return the value of the nearest pixel within the image for use in the convolution.

For convolution via Fourier transform the algorithm is significantly different as it includes no loops, this is one of the reasons for the increase in speed over the basic convolution.

Result: Returns the convolution between an image (I) and a kernel (K)
Pad(I) by the size of K;
Padkernel = Place K in first indexs of zeros(pad(I));
Compute fast Fourier transform of Pad(I) and Padkernel = fI, fK;
Compute element wise multiplication of fI and fK;
Compute the inverse Fourier transform of the result ;
Remove the padding;

Algorithm 2: Convolution via Fourier transform

2.2 Experimentation and Conclusions

The experimentation carried out in this section consists of two parts, the first is a comparison of my implementation and the MATLAB inbuilt implementation to check both compute the same result. The second is a comparison of the time required to compute the convolution method and the Fourier transform method.

Figure 1 shows the outputs of the implemented functions compared to the inbuilt functions. All implemented functions had zero Mean Squared Error (MSE) with the inbuilt functions. There is also zero error between Gaussian blurs between the Fourier and convolution methods.

When comparing the time taken for the convolution function and the Fourier method, a Gaussian blur kernel was used with standard deviation 3. The time was recorded with the following code:

Result: Returns the time taken for function (F)
Start clock;
Run F;
Stop clock;
t = elapsed time;
Return t;

Algorithm 3: Time Function

The convolution algorithm had a run time of 3.57 seconds and only 0.0216 seconds via the Fourier method. These times were averaged from the individual timings shown in table 1. The Fourier method in this case is over 160 times faster to compute and the zero MSE between the methods. This is due to the success of the convolution theorem and the reduced computational complexity of the element wise multiplication used in the Fourier method.

Convolution	Fourier
3.5966	0.0227
3.5813	0.0211
3.5966	0.0209
3.5688	0.0208
3.5349	0.0227
3.57	0.0216

Table 1: A table of times taken (in seconds) to compute a blur on an image via convolution and Fourier methods. The average is shown at the bottom of the table.

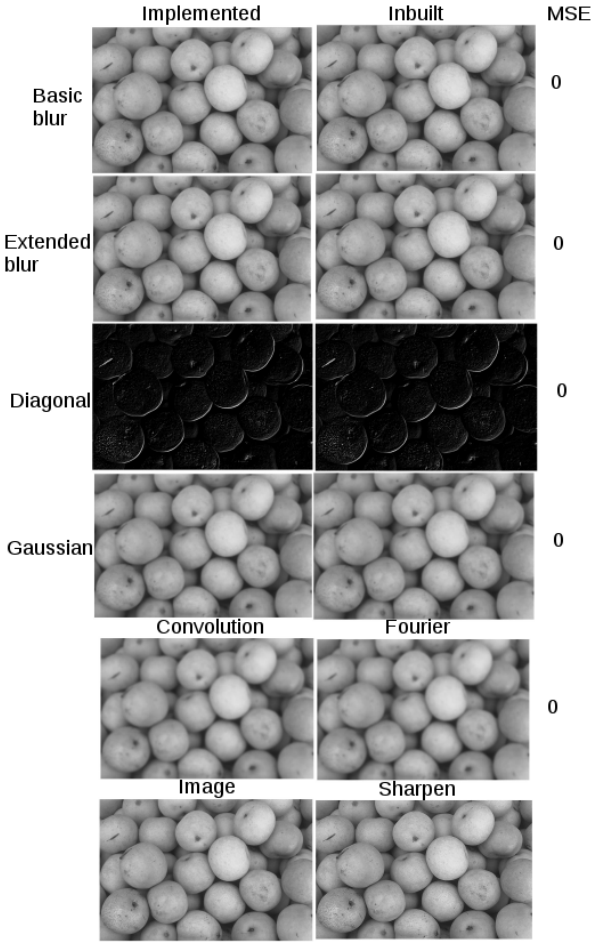

Figure 1: A comparison between images formed with the inbuilt MATLAB functions on the right and the implemented functions on the left. The error between the outputs is the Mean Squared Error (MSE). The bottom two rows show a comparison between Gaussian blurs by convolution and Fourier methods, as well as a comparison between the unedited image and the image sharpened by subtraction of blur.

To summarize this exercise on convolution, basic convolution as well as a method of handling image borders was implemented successfully with zero MSE compared to the inbuilt functions. Various image filters were also discussed and implemented, including Sobel filters, blur/sharpen, and Gaussian blur. Finally a Fourier method for convolution was discussed and implemented based on the convolution theorem. It was also shown that the Fourier method is significantly faster than basic convolution due to the reduction of computational complexity down to element wise matrix multiplication.

3 Features

3.1 Review of Features

This Exercise focuses on three different feature detection algorithms, Canny edge detection, Harris corner detection and Difference-of-Gaussian (DoG) interest point detection.

Edge detection for images encompasses a variety of methods that aim to identify points at which there is a discontinuity. The Canny edge detector, developed by John Canny in 1986 [4], is one of the most established methods of edge detection. The Canny edge detector can be broken down into five main steps; smoothing the image to reduce noise, computing intensity gradients, apply non-maximum suppression, threshold the result, and finally track edges by hysteresis.

Typically the smoothing is carried out by applying a Gaussian filter, the larger the Gaussian kernel used the lower the detectors sensitivity to noise. However, the more the image is blurred the lower the detectors sensitivity to high frequency edges. The intensity gradients can then be found using any edge detection operator, such as the Sobel operator discussed previously [1]. Both the horizontal,$\boldsymbol{G}_x$, and vertical, $\boldsymbol{G}_y$, gradients are required to calculate the total edge gradient map $\boldsymbol{G}$. The direction of each edge is then computed into an edge direction angle map, $\boldsymbol{\Theta}$, shown in equation 5. The angle for each pixel is rounded to either 0, 45, 90, or 135 degrees, representing vertical, horizontal and the two diagonals.

$$\begin{aligned} \boldsymbol{G} &= \sqrt{\boldsymbol{G}_x^2 + \boldsymbol{G}_y^2} \\ \boldsymbol{\Theta} &= atan2(\boldsymbol{G}_y, \boldsymbol{G}_x) \end{aligned} \tag{5}$$

Non-maximum suppression is used to thin each detected edge to one pixel as the current extracted edges are blurred. Non-maximum suppression functions by setting all other gradient values to zero other than the local maximum. This is done by comparing the value of the current pixel with the pixel in the positive and negative directions (as

specified by Θ for the pixel).The suppressed map of pixel edges are then passed through a threshold to filter out any particularly weak values. Both a high and low threshold are used, if a value is above the high threshold it is marked as a strong pixel, if it is above the weak it is marked as a weak pixel, otherwise it is removed. The final image is then processed by tracking edges via hysteresis. Here weak pixels are checked to see if they connect to a strong pixel, if so they are updatated to strong pixels, any unconnected pixels are removed and the final image is returned.

A commonly used corner detection operator is the Harris corner detector. Created by Chris Harris and Mike Stephens [5], it is based on Moravec's corner detector [6][7] but takes into account the direction and cornerness score to more accurately distinguish between edges and corners. A corner is defined as a local region which contains two intersecting different edge directions.

The Harris corner algorithm primarily focuses on calculating the structure tensor, M, of the image, I, as follows;

$$M = \sum_{(x,y)} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{6}$$

where the individual I_a components are the image filtered by a one dimensional gradient filter (e.g. $[-1, 0, 1]$ for the x component). The algorithm can be described in five steps; convert the image to grayscale, calculate the spatial derivatives I_x and I_y, create the structure tensor M, calculate the Harris corner response, carry out non-maximum suppression and return the results. The cornerness response, R, is calculated as

$$R = det(M) - k(trace(M))^2 \tag{7}$$

where k is a constant. The resultant value R for each pixel describes the cornerness of the pixel; close to zero is nothing, positive is a corner and negative is an edge.

The final feature detection algorithm to be discussed in this section is the difference of Gaussians (DoG) interest point detection algorithm. This method functions by enhancing features by using Gaussian kernels to calculate various blurred versions of the image and subtracting one from the other. This preserves the spatial information that lies between the two frequency bands that are suppressed by the blur, enhancing those features. The DoG can thus be imagined as a band-pass filter for images. DoG can be used for blob detection, and it is this application that will be explored here.

In blob detection, the DoG approach is used to approximate the Laplacian of Gaussian which can be computed as the limit of the difference between two Gaussian blurred images [8]. This is based on scale-space representation of the image, where each individual blurred version of the image will have different structures suppressed [9]. By taking local extrema over scales of different combinations of Gaussians, features can be detected [10].

In this implementation, the image is resized into a collection of images called octaves. Each octave has a selection of Gaussian blurs taken of various sigma values. Neighbouring blurs are then subtracted from each other to provide a similar effect as applying a Laplacian of a Gaussian. This enhances blobs of equivalent scale to the Laplacian. Local maxima, both in scale and location, are then found for each of the enhanced images and repeated maxima are removed. The detected points are then shown on the original image once scaled to the correct size of the Laplacian used to detect them.

3.1.1 Features Implementation

The Canny edge detection algorithm can be described in pseudocode as the following:

```
Result: Returns a binary output of the
        edges of an input image (I)
Convert I to grayscale;
Blur I by Gaussian filter;
Filter I to get G_x and G_y gradient maps;
Compute G ;
Compute Θ;
for Pixel in I do
    if Θ(pixel) rounds to 0 then
        if G(pixel) is local max in 0 degree
         direction then
            Store G(pixel) in EdgePixels
        end
    end
    Repeat for 45, 90, and 135 degrees;
end
for pixels in EdgePixels do
    if pixel > upper threshold then
        Mark pixel as Strong
    else if pixel > lower threshold then
        Mark pixel as Weak
end
for pixel in WeakPixel do
    if pixel is next to a StrongPixel then
        Mark pixel as Strong
    end
end
Return StrongPixels as binary map;
```

Algorithm 4: Canny edge detection

The Harris edge detection algorithm was imple-

mented as follows.

Result: Returns a binary map of corners of an input image I
Compute gradient maps I_x and I_y;
Compute structure tensor M;
for *pixel in I* **do**
  Compute cornerness score R;
  if $R < 0.3$ **then**
    Set R = 0 as not a corner
  end
end
Find local maxima of cornerness score;
Return binary map of corners;

Algorithm 5: Harris corner detection

The DoG blob detection algorithm can be summarized as:

Result: Returns a plot of scaled blobs on image I
Convert I to grayscale;
for *Octaves (o)* **do**
  Resize I by 0.5^o;
  for *Scales (z)* **do**
    Compute Gaussian blur of I;
  end
  Compute difference of neighboring blurred I (DoG's);
  Compute local minima/maxima of DoG's;
end
for o **do**
  for z **do**
    Check for repetition of minima/maxima between z scales;
    Remove smaller scale repete;
  end
end
for o **do**
  for z **do**
    Check for repetition between o octaves (pixels map 1-2 due to I resize between octaves);
    Remove lower octave repeats;
  end
end
Resize each scale map in each octave to original I size;
Plot each point on I as a circle with radius α octave and scale;

Algorithm 6: DoG blob detection

3.2 Experimentation and Conclusions

The experiments carried out in this section involve qualitatively comparing the implemented function to the MATLAB inbuilt function for Canny edge detection. For Harris corners and DoG blob detection a qualitative demonstration of the algorithm will be shown.

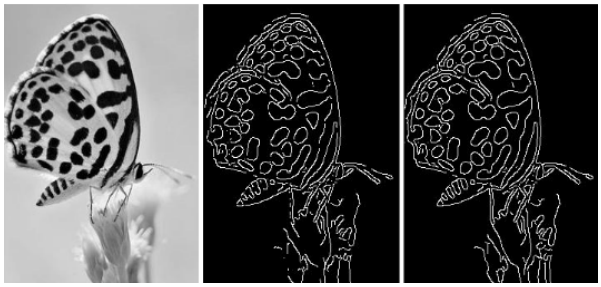

Figure 2: The output of Canny edge detection. The original image is on the left, the implemented algorithm output in the center, and the inbuilt algorithm output on the right.

As can be seen in figure 2, there is very minor differences between the implemented and inbuilt versions of the edge detection outputs. The minor differences are most likely due to slight differences in the lower threshold parameter as this is specified in the implementation but is calculated from the high threshold in the inbuilt function.

The output of the implemented Harris corners algorithm can be seen in 3. It is clear the algorithm is successfully detecting corners in the image. However, it can be seen that some corners are ignored such as those in shadow. This is most likely due to the accepting threshold for the cornerness score being too high, although decreasing this threshold starts to detect edge features.

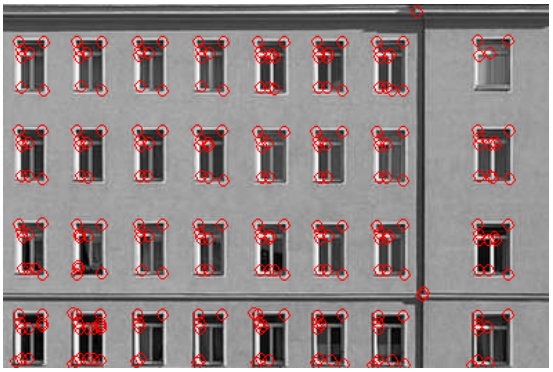

Figure 3: The output of the implemented Harris corner algorithm. Detected corners are marked with red circles on the original image.

The DoG algorithm can be used to detect both maxima and minima, white or black blobs in a grayscale image respectively. Figure 4 demonstrates the functionality of the implemented DoG algorithm both on maxima and minima of various scales. It also successfully marks each feature with

an appropriately sized marker which is proportional in size to the Laplacian used for the identification of the feature.

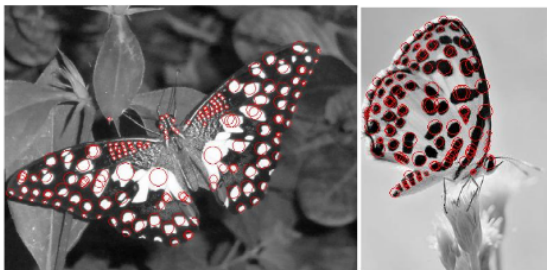

Figure 4: The output of the implemented DoG blob detection algorithm, demonstrating both maxima and minima detection (left and right respectively).

The implementations of all of these algorithms could be improved as computational performance for all is poor on large, or high resolution, images. This is due to the functions being coded with multiple nested loops. It is possible to optimise these by, perhaps, vectorizing many of the calculations and removing some of the loops.

4 Matching

4.1 Review of Matching

This section focuses on feature matching between images and the subsequent image stitching that can be accomplished with it. First a brute force feature matching algorithm will be implemented, followed by a function to estimate homography. This homography estimate will then be used to warp and aline one image with another. The homography estimate will then be improved by implementing a RANSAC model.

A brute force feature matching technique finds corresponding pairs of features by comparing descriptor distances. Features can be detected by algorithms such as SURF [11], which will be used in this implementation, and then compared between images by measuring the euclidean distance between every feature on each image. The shortest distance pairs for each image are recorded and only pairs that have matching shortest distances are kept. To improve matching and filter out non-unique features the ratio test is applied. The ratio test, as described by Lowe [12], involves also recording the second closest neighbor in the feature matching phase. This second closest match can provide an estimate of the density of false matches within the feature space. As shown in figure 5, Lowe finds that rejecting all matches in which a distance ratio of 0.8 or more eliminates 90% of false patches. In this implementation a value of 0.6 was chosen. The remaining matched feature points can be plotted on both input images to show correspondances.

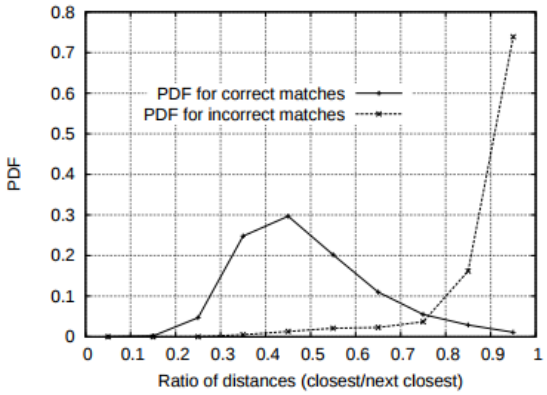

Figure 5: The probability distribution function of a correct match (PDF) plotted against the ratio of distances. The solid line shows correct matches, the dotted line incorrect matches. [12]

Any two images in of the same surface are related by a homography [13]. Homography can be represented as a matrix H;

$$H = R - \frac{tn^T}{d} \tag{8}$$

where R is the rotation matrix in which the second image is rotated in relation to the first, t is the translation vector for the same image, n is the normal vector of the plane, and d is the distance to the plane from the view point. The homography matrix can be estimated by using matched feature points. In this implementation four well matched features are chosen from the SURF matched features between two images that make up a panorama. Each time the matrix is computed, a minimum of four pairs of points are used. These points are arranged into a matrix for each pair of points;

$$p(i) = \begin{bmatrix} -x_{in}(i) & -y_{in}(i) & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -x_{in}(i) & -y_{in}(i) & -1 \\ x_{in}(i)*x_{out}(i) & y_{in}(i)*x_{out}(i) & x_{out}(i) & & & \\ x_{in}(i)*y_{out}(i) & y_{in}(i)*y_{out}(i) & y_{out}(i) & & & \end{bmatrix} \tag{9}$$

where x and y indicate the coordinates of the match point, i, for the input and output images. single value decomposition is carried out on each of the matrices to provide the V unitary matrix which provides each of the entries of the homography matrix. Warping one image onto the other using the homography matrix is then a fairly trivial task, simply transform the second image by the homography matrix and output the result onto the first image.

The feature matching can be improved further by passing the matched points from SURF through a RANSAC (random sample consensus) algorithm. First proposed by Fishler and Bolles in 1981 [14],

the assumption of the method is that the data consists of inliers. The inliers distribution can be explained by some model, as such the probability of choosing inliers is higher than choosing outliers; the RANSAC algorithm exploits this by making a prediction of the distribution by choosing only the inliers. This does rely on the number of inliers being sufficiently high, however.

The RANSAC algorithm is composed of two steps that are repeated iteratively:

- Randomly select a sample subset of minimal data points. Compute a fitting model and parameters from this sample.
- Check all other data points with the computed model. Any point which does not fit the model will be considered an outlier. If sufficient points fit the model the model is considered successful.

This process is iterated a fixed number of times, either accepting or rejecting the model formed. The model with the most inliers is then chosen to be used for the homography estimation.

4.1.1 Matching Implementation

Brute force feature matching was implemented by first extracting features via SURF and then passing the points and features into the following function.

Result: Returns matched pairs of features between images I1 and I2.
for *Each Feature in I1 and I2* **do**
  Measure distance between F1 and F2;
end
for *Features in I1 and I2* **do**
  Get shortest and second shortest distance pairs;
end
Compare shortest pairs for I1 and I2, keep only those where the shortest in I1 matches I2;
Perform ratio test for feature pairs;
Return matched features ;

Algorithm 7: Brute force feature matching

The homography function takes in four matched pairs of points and functions as follows.

Result: Returns the homography matrix, H, from four matching feature points
for *Each pair of points* **do**
  Compute matrix p(i) via equation 9;
end
Vertically concatenate p(i) into P;
Single value decomposition of P to return V;
Arange V into 3x3 matrix H;

Algorithm 8: Homography estimation

Mapping points from image one to image two is then done by multiplying the second image by H - this warps the image to the same perspective as the first image. It is regularly the case that the output point from this multiplication lies between multiple pixels, in this case bilinear interpolation is used.

Result: Returns the interpolated value between four pixels, matrix I, which is distance (a, b) from pixel I_{11}.
$f_1 = (1 - a) * I_{11} + a * I_{12}$;
$f_2 = (1 - a) * I_{21} + b * I_{22}$;
$f = (1 - b) * f_1 + b * f_2$;
Return value f;

Algorithm 9: Bilinear interpolation

The RANSAC algorithm takes in feature points from two images and the following parameters; minimum number of points used for consensus n, number of iterations k, inlier threshold t.

Result: Returns a homography matrix H the set with most inliers
for *k iterations* **do**
  Randomly select n sample pairs;
  Perform homography estimate on set of random pairs;
  Compute distance between all points;
  if *distance* $> t$ **then**
    Refit estimated homography
  end
end
Return H for set with most inliers;

Algorithm 10: RANSAC

4.2 Experimentation and Conclusions

For the brute force feature matching as well as the homography calculation, two images that contain an overlapping region where used. The images contain many similar features, in particular similar arches to make the problem more challenging. The implemented brute force algorithm is compared to the inbuilt SURF algorithm in figure 6. It can be seen that identical features are identified in both, proving the brute force algorithm functions as intended.

For verifying the functionality of the homography estimation, four well matched points were taken for the homography calculation. A point was taken in the first image and its equivalent position in the second image was computed using the homography estimate. It is shown in figure 7 that the point corresponds well between the two images, verifying the accuracy of the homography estimation. The estimation is then used to warp the second image and overlay it on the first to create the panorama, also shown in figure 7.

Figure 6: The inbuilt SURF feature matcher (top) and the implemented brute force matcher (bottom). Matched features are marked with red circles and green crosses and connected by yellow lines.

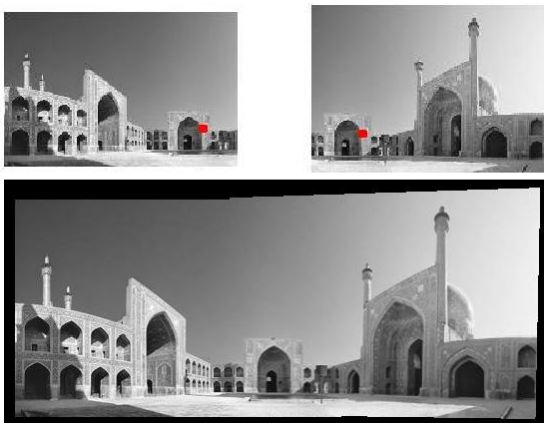

Figure 7: The top image shows a randomly selected point in the left hand panel and its corresponding point, as computed by the homography estimation, in the right panel. The bottom image shows the result when the second image is warped by the homography matrix and overlaid on the first creating a panorama.

To challenge the RANSAC implementation, a higher resolution image with many more similar visual features was used. Figure 8 shows the panorama output of the RANSAC algorithm showing well matched features and a seamless image stitch.

Figure 8: A high resolution panorama created by RANSAC homography estimation.

5 Camera Calibration

5.1 Review of Camera Calibration

Based heavily on Zhang's technique for camera calibration [15], the implemented technique requires only a planar pattern viewed different orientations. For this implementation, four sets of corresponding points are needed. Calibration requires knowing the intrinsic parameters and extrinsic parameters of the camera. The extrinsic parameters require estimating the rotation matrix, R, and the translation matrix, t. The intrinsic parameters are bundled into the intrinsic matrix,

$$A = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{10}$$

where (u_0, v_0) are the coordinates of the principle point, α and β are scale factors in the image u and v axis, and γ is a parameter describing the skewness of the two image axis.

Firstly the homographies are estimated between the corresponding points, the same technique is used as in section 4.1 for estimating the homography matrix H. The intrinsic matrix, A, can then be computed from H by using the following closed-form solution.

$$B = A^{-T}A^{-1} = \begin{bmatrix} \frac{1}{\alpha^2} & -\frac{\gamma}{\alpha^2\beta} & \frac{v_0\gamma - u_0\beta}{\alpha^2\beta} \\ -\frac{\gamma}{\alpha^2\beta} & \frac{\gamma^2}{\alpha^2\beta^2} + \frac{1}{\beta^2} & -\frac{\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} + \frac{v_0}{\beta^2} \\ \frac{v_0\gamma - u_0\beta}{\alpha^2\beta} & -\frac{\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} & \frac{(v_0\gamma - u_0\beta)^2}{\alpha^2\beta^2} + \frac{v_0^2}{\beta^2} + 1 \end{bmatrix} = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{12} & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{bmatrix} \tag{11}$$

As shown, B is symmetric and can be defined by the following 6D vector:

$$b = [B_{11}, B_{12}, B_{22}, B_{13}, B_{23}, B_{33}]^T \tag{12}$$

By defining the i^{th} column of H to be $h_i = [h_{i1}, h_{i2}, h_{i3}]^T$, we can form $h_i^T B h_j = v_{ij}^T b$ with

$$\begin{aligned} v_{ij} = &[h_{i1}h_{j1}, h_{i1}h_{j2} + h_{i2}h_{j1}, \\ &h_{i2}h_{j2}, h_{i3}h_{j1} + hi1h_{j3}, \\ &h_{i3}h_{j2} + hi2h_{j3}, h_{i3}hj3]^T \end{aligned} \tag{13}$$

This allows us to solve, via single value decomposition, the following equation for b from only knowing H.

$$\begin{bmatrix} v_{12}^T \\ (v_{11} - v_{22})^T \end{bmatrix} b = 0 \tag{14}$$

Once b is known, B can be formed and, via rearrangement of equation 11, all the components of A can be found.

From A the extrinsic parameters can be computed via

$$\begin{aligned} r_1 &= \lambda A^{-1} h_1 \\ r_2 &= \lambda A^{-1} h_2 \\ r_3 &= r_1 \times r_2 \\ t &= \lambda A^{-1} h_3 \end{aligned} \tag{15}$$

with $\lambda = 1/||A^{-1}h_1$. The rotation matrix is then formed from the row vector of r.

5.1.1 Camera Calibration Implementation

The implementation of estimating the intrinsic and extrinsic properties of a camera is broken into three functions. The first is the overarching algorithm:

Result: Returns intrinsic parameter matrix, A, for four sets of corresponding points from four images.
for *Each image* **do**
  Estimate Homographies (see algorithm 8);
end
Estimate Intrinsic parameters (see algorithm 12);
Estimate Extrinsic parameters using equation 15;

Algorithm 11: Camera Calibration

The intrinsic parameters are computed via a separate function:

Result: Returns A given a set of homographies
for *Each H* **do**
  Compute v_{ij};
  Compute $[v_{12}^T, (v_{11} - v_{22})^T]^T$;
end
Solve single value decomposition to find b;
Form B from b;
Compute intrinsic parameters α, β, γ, v_0, and u_0 from B using equation 11;
Arrange A as in equation 10;
Return A;

Algorithm 12: Intrinsic parameter estimation

5.2 Experimentation and Conclusions

For the images taken in the following section, a 12-megapixel smartphone camera was used. For calibration, four images of a checkered grid pattern were taken at different angles. The calibration images used can be seen in figure 9 as well as the points used. Four points from each of the four images were used, the real world measurements were taken from the point indicated by the red circle on each of the images as the real world origin. The real world positions are measured in centimeters and can be seen in table 2 along with the image coordinates.

Point	Real	image 1
1	(0,0)	(1040,1651)
2	(0,2.9)	(1036,1398)
3	(3.2,2.9)	(1314,1392)
4	(3.2,0)	(1319,1638)

image 2	image 3	image 4
(1192,1663)	(1552,1916)	(805,1811)
(1186,1421)	(1410,1781)	(768.6,1594)
(1446,1408)	(1557,1622)	(985.1,1560)
(1451,1642)	(1699,1755)	(1022,1783)

Table 2: The real world measurements (in centimeters) and image coordinates (in pixels) for the four calibration images and four chosen corresponding points. Coordinates are given as (x, y). The calibration images can be seen in figure 9

The computed intrinsic matrix, A, for this camera from these four calibration images was found to be

$$A = \begin{bmatrix} 1.36 \times 10^2 & 0.74236 & 1.12 \times 10^3 \\ 0 & 0.01159 & -16.8737 \\ 0 & 0 & 1 \end{bmatrix} \tag{16}$$

which can be deconstructed to give the individual intrinsic parameters as

$$\begin{aligned} \alpha &= 1.36 \times 10^2 \\ \gamma &= 0.74236 \\ u_0 &= 1.12 \times 10^3 \\ \beta &= 0.01159 \\ v_0 &= -16.8737 \end{aligned} \tag{17}$$

The calibration could potentially be improved by using multiple calibration points and comparing the computed intrinsic matrices for differences to test consistency in the estimation.

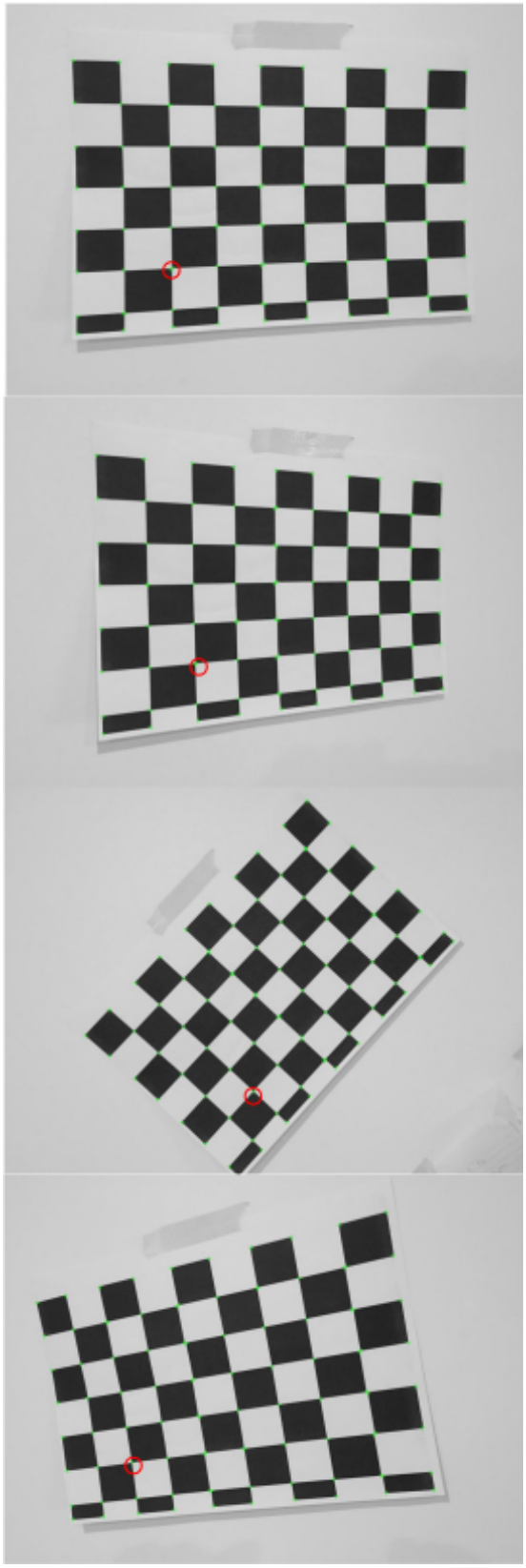

Figure 9: The four images used for camera calibration. The real world origin point is shown with a red circle. The Harris corners are shown with green crosses. The chosen calibration points are the corners on the origin, directly above, to the right, and above right of the origin.

6 Fundamental Matrix Estimation

6.1 Review of Fundamental Matrix Estimation

The fundamental matrix, F, contains the information to describe the projective relation of two cameras [16]. F can be computed from pixel coordinates of corresponding points within uncalibrated images and is defined by

$$x'^T F x = 0 \tag{18}$$

where x and x' are homogeneous coordinates of corresponding points in stereo images. The epipolar line Fx describes a line which x' lies on in the other image.

The essential matrix, E, is the specific case of the fundamental matrix in the case of normalize image coordinates. As such it can be computed from F given the calibration matrices of the cameras (K and K');

$$E = K'^T F K \tag{19}$$

E is also defined as

$$E = R[t]_\times \tag{20}$$

where $[t]_\times$ is the matrix representation of the cross product with the transformation matrix t. The essential matrix directly describes the extrinsic properties of the pair of cameras, as shown by equation 20.

The fundamental matrix is computed in this implementation via the eight-point algorithm [17]. This requires eight corresponding image points which may, or may not, be normalised (however it was found to be more stable with normalised points [18]). These normalised points are formed by translating the original points to a new origin and scaled to a mean distance from the origin of $\sqrt{2}$. From these points a constraints vector is formed,

$$C = [x'_1 x_1, x'_1 x_2, x'_2, x'_2 x_1, x'_2 x_2, x'_2, x_1 x_2, 1] \tag{21}$$

where x and x' are the image coordinates of the form $x = [x_1, x_2, 1]^T$. This allows a homogeneous linear equation to be formed between C and a nine dimensional vector of the components of F. Here it follows that single value decomposition can be used to calculate the orthogonal matrix V which is reshaped to form an unconstrained F. Single value decomposition can be employed again, this time on the newly found, rank two matrix, F to apply the constraints. This is done by first reconstructing the

two largest singular values from the diagonal matrix, D, from the single value decomposition into a new 3×3 diagonal matrix, S. F is then given as

$$F = USV^T \tag{22}$$

Finally the normalisation is removed by applying the inverse of the original normalisation transformation, returning the finalised fundamental matrix.

The Essential matrix can then be computed from the calibration matrices as shown in equation 19. The rotation and translation matrices, R and t, can be found by decomposing E via single value decomposition;

$$E = UDV^T \tag{23}$$

The diagonal matrix, D, contains the singular values of E. Due to the constraints on E there must be two identical singular entries and one zero value. As such we can define a new matrix

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{24}$$

With this we can establish the following equations for R and $[t]_\times$.

$$\begin{aligned} [t]_\times &= UWDU^T \\ R &= UW^{-1}V^T \end{aligned} \tag{25}$$

where $[t]_\times$ is the cross product representation of the transformation matrix.

6.1.1 Fundamental Matrix Estimation Implementation

The basic implementation of the fundamental matrix estimation via the eight-point algorithm is as follows:

Result: Returns fundamental matrix, F, given a set of at least 3×8 homogeneous points
Normalise the points using algorithm 14;
Form the constraints vector C;
Compute singular value decomposition of C;
Form a preliminary F matrix from the single value decomposition output V;
Compute singular value decomposition on preliminary F;
Form the constrained F as in equation 22;
Remove normalisation as $F = T_2^T * F * T_1$ where T_x is the transformation matrix returned from algorithm 14;

Algorithm 13: Fundamental matrix estimation via 8-point algorithm

The normalisation algorithm used in the eight-point algorithm is as follows:

Result: Normalises 2D homogeneous points and returns the transformation used
Check points are homogenous, if not make them homogenous;
Compute the centroid;
Position the new origin point using the centroid;
Measure the distance between points and return the average distance;
Compute a $\sqrt{2}$ scale factor;
Create a transformation matrix using the scale factor and centroid;
Transform the input points by the transformation matrix;
Return normalised points and transformation matrix;

Algorithm 14: Normalise 2D points

6.2 Experimentation and Conclusions

For the basic implementation of the fundamental matrix calculation two images where used, see figure 10, from two phone cameras taken adjacent to each other. Nine corresponding points were chosen manually from the Harris features with the following homogeneous coordinates:

Image 1	Image 2
(604.2, 483.3)	(449.3, 536.7)
(601.1, 448.8)	(441.6, 499.4)
(561.4, 423.1)	(388.9, 472.2)
(699.1, 319.7)	(542.9, 356.1)
(656.8, 354.2)	(496.6, 394.7)
(570.1, 380.9)	(394.1, 424.3)
(548.6, 243.1)	(364.1, 266)
(652, 203)	(495.5, 221.9)

Table 3: The 8 chosen corresponding homogeneous coordinates from the two images in figure 10

From these points the fundamental matrix was computed as

$$F = \begin{bmatrix} 1.31 \times 10^{-5} & 5.87 \times 10^{-5} & -0.0290 \\ -6.26 \times 10^{-5} & 5.37 \times 10^{-5} & 0.0191 \\ 0.0141 & -0.0508 & 9.5064 \end{bmatrix} \tag{26}$$

Figure 10: Two images used for computation of the fundamental matrix shown in equation 26. The Harris features are shown with green crosses, those chosen for calculation of F are shown in table 3.

For the addition of RANSAC in the selection of the best corresponding points to use for estimating F, two different images where used which can be seen in figure 11. Here the corresponding points where found via SURF matched features. These points where then passed through a RANSAC algorithm to return the best fundamental matrix.

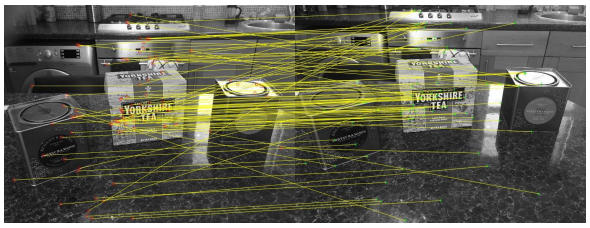

Figure 11: The images used for the RANSAC estimation of the fundamental matrix. The matched SURF features are shown.

The F matrix returned by the RANSAC method is as follows:

$$F = \begin{bmatrix} 5.38 \times 10^{-9} & -1.16 \times 10^{-9} & -4.69 \times 10^{-6} \\ 2.63 \times 10^{-10} & 9.12 \times 10^{-9} & -1.52 \times 10^{-5} \\ -6.48 \times 10^{-6} & -7.86 \times 10^{-6} & 0.02077 \end{bmatrix} \tag{27}$$

The RANSAC estimation of the fundamental matrix should be superior is it uses many corresponding points and multiple iterations to find the solution of the most inliers and thus the smallest error. The RANSAC implementation calculated the fundamental for two images take with the same camera at slightly different locations. Originally two images from the stereo camera setup where going to be used, however, it proved to be very hard to get a large amount of corresponding points required for the RANSAC algorithm so to prove the functionality the images in 11 were used. As such the fundamental matrix calculated before is not expected to be in anyway similar to the RANSAC computed matrix.

The essential matrix between the two camera positions for image 11 can now be found. The intrinsic matrix for the camera is known (equation 16). As the same camera is used for both images the same intrinsic matrix is used for both. The fundamental matrix is also know for the two camera positions (equation 27) so the essential matrix can be found. It was computed to be:

$$E = \begin{bmatrix} [9.94 \times 10^{-5} & 5.41 \times 10^{-7} & 1.88 \times 10^{-4} \\ 5.43 \times 10^{-7} & 2.96 \times 10^{-9} & 8.51 \times 10^{-7} \\ -5.98 \times 10^{-5} & -4.35 \times 10^{-7} & 0.01542 \end{bmatrix} \tag{28}$$

From E the transformation matrix, t, and the rotation matrix, R, which describe the extrinsic differences between the two camera positions can be found as

$$t = \begin{bmatrix} -8.42 \times 10^{-5} & -0.01542 & 8.42 \times 10^{-7} \end{bmatrix}$$
$$R = \begin{bmatrix} 0.00166 & -0.99998 & 0.00670 \\ -0.10000 & -0.00166 & 1.83 \times 10^{-5} \\ 7.13 \times 10^{-6} & 0.00670 & 0.99998 \end{bmatrix} \tag{29}$$

The accuracy in the estimation for E is potentially questionable due to the small amount of corresponding points found in the test images. Multiple test images were used with multiple degrees of rotation and translation however it consistently proved difficult to get a large number of matched points. The final test image used still had a limited number of matched points as well as many points which are clearly not accurate correspondances. It is highly possible that this will negatively affect the estimate. As this estimate for E is used in the following section, the final output of that section is likely to also show errors which may be the result of a poor estimate of the essential matrix.

7 Triangulation of 3D Points

7.1 Review of Triangulation Of 3D Points

Referring to the process of determining a points position in three dimensional space given by its two dimensional projections, triangulation can be carried out with the camera data we have so far computed. Corresponding points in two images draw a line into the image plane in three dimensional space, where these two lines intersect is the location of the point. Computing this is based on the epipolar geometry of the two cameras, fortunately the majority of what is needed to solve the problem has already been found via the camera calibration and the fundamental matrix.

To triangulate pairs of homogeneous points, x_1 and x_2, an object known as the projection matrix for each camera must be known. The projection

matrix, P, is a 3×4 matrix which describes the mapping of real world, three dimensional points, into the two dimensional image plane of the camera. We can describe the projection matrix of the second image as a relation to the first. In this case P_1 becomes

$$P_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = (I|0) \tag{30}$$

The second part of the equation represents the concatenated form. The cameras calibration is applied to the projection to give the finalised P_1 as

$$P_1 = K_1(I|0) \tag{31}$$

The projection matrix for the second camera is more complex to compute. It is found by decomposing the essential matrix, E, that was found earlier for the two cameras into the rotation and transform matrices. The decomposition leads to four possible projection matrices. The possible results are due to how the projection is taken, R is found in relation to the plane W where W is defined as

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{32}$$

This leads to two possible R values, one for W and one for W^T. Similarly t is found from the third column of the decompositions unitary matrix, U, and could either be positive or negative. P_2 is then a concatenation of these too matrices as $(R|t)$. This leads to four possible combinations.

To find the correct matrix a test corresponding point pair is used, the depth coordinate, z, is computed for the first camera using a normalised P_1 and the second camera using the various normalised potential P_2's. The combination that has positive z for both points (as the point will exist into the plane based on the set up and coordinate system in use) will be chosen.

The second cameras projection matrix will also have the camera calibration applied in the same way as it was for P_1 in equation 31. Now both projection matrices are known, each of the corresponding point pairs can undergo triangulation to calculate their position in three dimensional space. This is achieved via linear triangulation methods as described by Hartley and Zisserman [19]. Here the linear components of the projected point, X, are used to form the matrix A, where $AX = 0$, as follows:

$$A = \begin{bmatrix} xP_1^3 - P_1^1 \\ yP_1^3 - P_1^2 \\ x'P_2^3 - P_2^1 \\ y'P_2^3 - P_2^2 \end{bmatrix} \tag{33}$$

where x and x' are the corresponding x-coordinates, y and y' the corresponding y-coordinates, and the superscript numbers on the projection matrices indicates the row of the matrix.

The triangulated point can then be solved via performing single value decomposition of A. The smallest singular value is used as a vector for the new coordinates. This is described in greater detail in Hartley and Zisserman's discussion on homogeneous methods for solving A [19]. The result is a unit singular vector for each point in three dimensional space, the full collection of points forms a point cloud when plotted.

7.1.1 Triangulation Of 3D Points Implementation

The triangulation implementation was divided into four functions. The first is the overall triangulation algorithm:

Result: Using two images from calibrated cameras with the essential matrix, normalised 3D coordinates are returned for corresponding points.
Import two images;
Import E, K_1, and K_2;
Find corresponding SURF feature points;
Get four possible P_2 matrices via algorithm 16;
Create P_1 as in equation 31;
Find correct P_2 via algorithm 17;
Compute 3D coordinates for each corresponding point pair via algorithm 18;
Plot points in 3D;

Algorithm 15: Triangulation

The overall algorithm refers to three others. The first is an algorithm for computing the possible projection matrices:

Result: Computes the 4 possible projection matrices from the essential matrix, E.
Perform single value decomposition of E providing unitary matrices U and V;
Create matrix for the plane W;
Compute $R_1 = UWV$ and $R_2 = UW^TV$;
Compute $t_1 = U_{3rd}$ and $t_2 = -U_{3rd}$ Form the 4 concatenation combinations of R_x and t_x;
Return the 4 potential P's;

Algorithm 16: Get possible projection matrices

The second, an algorithm for computing which

is the correct projection matrix:

Result: Returns the correct P_2 from the 4 potential projection matrices by using a corresponding point test pair
Define $P_1 = (I|0)$;
Apply camera calibration to the test points;
for *Each potential* P_2 **do**
  Project test points in 3D using A from equation 33;
  Normalise A;
  Perform single value decomposition of A to get 3D coordinate of test point;
  Apply potential P_2 and P_1 to the 3D point to get the depth in the projection, z;
end
Check which z of the corresponding points is positive for both projections and return the respective P_2;

Algorithm 17: Find correct projection matrix

The final algorithm is for computing each points position in three dimensions using the appropriate projection.

Result: Outputs the 3D coordinates for corresponding points using their appropriate projections.
for *Each point pair* **do**
  Compute A as in equation 33;
  Perform single value decomposition of A;
  Return the smallest singular value which is a vector of the 3D coordinates;
end
Return all projected points;

Algorithm 18: Triangulate coordinates

7.2 Experimentation and Conclusions

The images used for triangulation of points were taken using one camera at two positions. The images are the same as those in 11, however only sixty of the strongest matched SURF features were used as points; the matched features can be seen in figure 12.

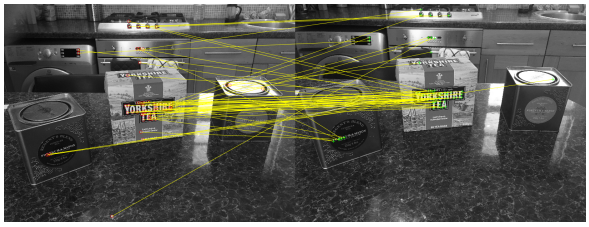

Figure 12: The matched features used as the corresponding points for the triangulation.

The essential matrix used is the one found via the RANSAC method shown in equation 28, and the camera calibration matrix used was the one shown in equation 16. The final normalized point cloud that was computed is shown in figure 13.

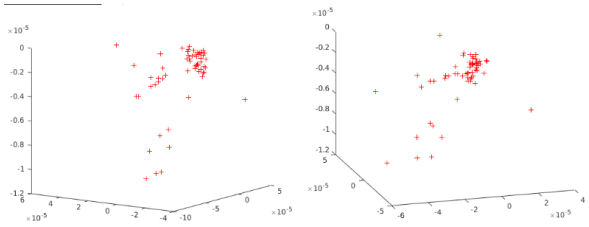

Figure 13: A 3D point cloud for the computed triangulated 3D points from the corresponding points in figure 12.

The points produced are in 3D as shown in the figure above. The tightly clustered points in a plane correspond to the clustered corresponding points in the center of the two images which are also on a flat surface. It is clear however that other points appear to be scattered randomly in the three dimensional space. As can be seen in figure 12, many points were detected as corresponding when they clearly are not, I believe these poor correspondances are responsible for the seemingly random scattered three dimensional points. The algorithm was tested with many test images, however it proved to be very difficult to get images with good feature correspondences, especially when creating dramatic rotation or translations between camera positions. This was using the inbuilt MATLAB SURF feature matcher.

The output of this algorithm could be greatly improved if the fundamental estimation was computed along side the triangulation and only the inliers of the RANSAC algorithm where used as the corresponding points for the triangulation. The result could also potentially be improved if multiple images where used and the triangulation computed from multiple projections into the three dimensional space.

References

[1] Irwin Sobel. An isotropic 3x3 image gradient operator. Feb 2014. Presentation at Stanford A.I. Project 1968.

[2] George C. Stockman Linda G. Shapiro. *Computer Vision*. Prentice Hall, Upper Saddle River, N.J. ; London, 2001.

[3] Arfken G. *Mathematical Methods for Physicists*. Academic Press, Orlando, FL, 1985.

[4] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, Nov 1986.

[5] Chris Harris and Mike Stephens. A combined corner and edge detector. page 50, 01 1988.

[6] Hans Moravec. Towards automatic visual obstacle avoidance. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, August 1977.

[7] Nilanjana Barman Debolina Das Subhabrata Chakraborty Nilanjan Dey, Pradipti Nandi. A comparative study between moravec and harris corner detection of noisy images using adaptive wavelet thresholding technique. *International Journal of Engineering Research and Applications*, 2:599–606, Sep 2012.

[8] Tony Lindeberg. Image matching using generalized scale-space interest points. *Journal of Mathematical Imaging and Vision*, 52(1):3–36, May 2015.

[9] Tony Lindeberg. *Scale-Space Theory in Computer Vision*. Springer, US, 1994.

[10] Tony Lindeberg. Feature detection with automatic scale selection. *International Journal of Computer Vision*, 30(2):79–116, Nov 1998.

[11] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. *Computer vision–ECCV 2006*, pages 404–417, 2006.

[12] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[13] Elan Dubrofsky. Homography estimation. *Diplomová práce. Vancouver: Univerzita Britské Kolumbie*, 2009.

[14] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[15] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22(11):1330–1334, 2000.

[16] Quan-Tuan Luong and Olivier D. Faugeras. The fundamental matrix: Theory, algorithms, and stability analysis. *International Journal of Computer Vision*, 17(1):43–75, Jan 1996.

[17] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133 EP –, Sep 1981.

[18] Richard I Hartley. In defense of the eight-point algorithm. *IEEE Transactions on pattern analysis and machine intelligence*, 19(6):580–593, 1997.

[19] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, 2 edition, 2003.