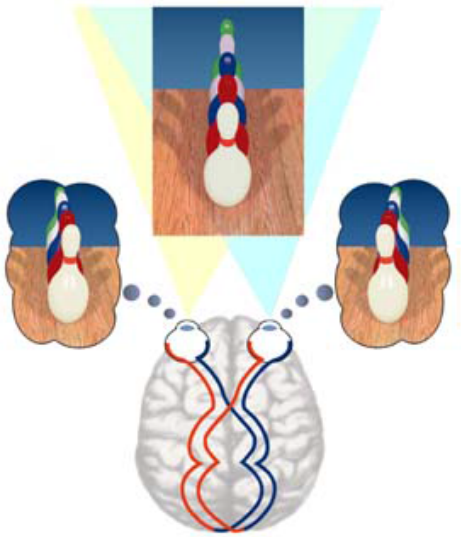
Final Report

The Art of Scientific Computing: Stereo Vision Project

Author:
Yitong Chen

Subject Handbook code:
COMP-90072



Faculty of Science
The University of Melbourne

24th May 2018

Subject co-ordinators: A/Prof. Roger Rassool & Prof. Harry Quiney

Contents

Chapter 1

Introduction

Most animals have two eyes, which could provide them with a much wider field of view. As for humans, they have a maximum horizontal view of approximately 190. Besides, about 120 degrees of 190 are seen by both eyes, thus the more important reason is overlapping of binocular vision could perceive a single three-dimensional image of the surroundings, that is referred to as stereopsis [4].

In computer stereo vision, the cameras could work as human's eyes, capturing two different views of the same scene, the relative depth information can be calculated in the form of a disparity map by comparing two images, which encodes the difference in horizontal coordinates of corresponding image points. The values in the disparity map are inversely proportional to the scene depth at the corresponding pixel location. But comparison is a challenge to computers. There are some techniques and assumptions to figure out the computational challenges. For example, "smoothness" is a measure that can be used to compare images, the assumption is that objects are more likely to be colored with a small number of colors, if we detect two pixels with the same color they most likely belong to the same object. As we live in a world with this property, human vision system seems to have evolved to use smoothness to interpret the world as well.

This project involves understanding and implementing Cross-Correlation, template matching (Gaussian fitting) in a noisy environment, calibration of images and exploring the technology into reality. MATLAB is a good tool for implementation, which has good built-in functions for cross-correlation and good ability for plotting, etc. We will talk about the cross-correlation in the first two chapters, and then ultilise and extend the algorithms to achieve a calibration model which mimics the depth understanding of a human. It will be first tested with ability to accurately analyse and represent depth information with stereo image pairs from resource library provided by the Univeristy of Melbourne, and then with the stereo photos of real world taken by people, finally optimise the system to improve accuracy as good as possible. More details will come in the following chapters.

Chapter 2

Cross-Correlations in one dimension

A good way to begin is getting familiar with cross-correlation. In signal processing, cross-correlation is a measure of similarity of two series as a function of the displacement of one relative to the other. This is also known as a sliding dot product or sliding inner-product, which is commonly used for searching a shorter template in the long signal. In our project, it could be used to find useful information in the visual images.

2.1 Cross-Correlation and Normalisation in 1D

The equation of one-dimensional spatial cross-correlation is shown below, where f and g are the compared signals here.

$$r = \frac{1}{N}\sum_{i=N}^{i=1}(f(i)-\bar{f})(g(i)-\bar{g})$$

Besides, the more general way in application is to implement a normalised cross-correlation as:

$$R = \frac{1}{N}\sum_{i=N}^{i=1}\frac{(f(i)-\bar{f})(g(i)-\bar{g})}{\sigma f \sigma g}$$

$$\sigma f = \sqrt{\frac{1}{N}\sum_{i=N}^{i=1}(f(i)-\bar{f})^2}$$

$$\sigma g = \sqrt{\frac{1}{N}\sum_{i=N}^{i=1}(g(i)-\bar{g})^2}$$

See we have two series of vectors, the idea is like sliding one from the beginning of the other to its end. The peak is the most similar part of two vectors, which would be the template we look for. Though it would be fast and convenient to use built-in cross-correlation functions *xcorr*, implementing the function manually would be very helpful for better understanding. The benefit of normalization is to reduce and eliminate data redundancy, making the result more compact, which means the shape of unnormalised vector and normalised vector should be same whereas the latter is normalised between 0-1.

It could be implemented in MATLAB by taking two vectors of the same size and passing one over the other, the steps employed to complete the implementation involved:

1. Generate a random vector.
2. Make a random vector of the same size.
3. Put same size of zero vectors at the beginning and end of the second vector, means that it has no correlation with two vectors at the beginning and the end.
4. Initialize the cross-correlation vector with size of length difference of two vectors plus 1.
5. Use for-loop to pass the shorter vector over the longer one, which could get the cross-correlation vector.

6. The normalisation of the vector is first divide two vectors by each own Euclidean norm, and then have same process as the previous cross-correlation.
7. Computer code see appendix A.1.

For example, we can generate the vector of size 3, and make the plots of the cross-correlation vector (figure 2.1) and the normalised cross-correlation vector (figure 2.2).

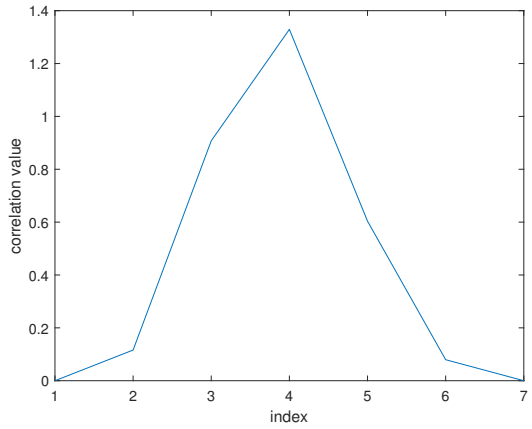


Figure 2.1: Cross-Correlation vector

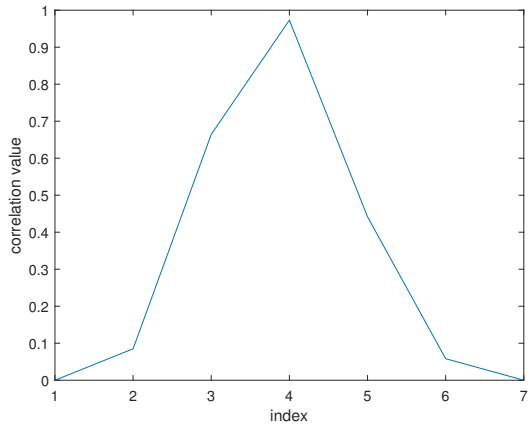


Figure 2.2: Normalised Cross-Correlation vector

The x-axis means index, while the y-axis means cross-correlation value in the figures. The curve in the first image (figure 2.1) has the peak at value about 1.4, which means the cross-correlation value of two vectors is about 1.4, when the signal is slid to index 4 of another. Similarly with the peak near 1.0 in the second image (figure 2.2), the range of cross-correlation value is normalised between zero to one.

2.2 Finding signal offset

Cross-correlation could be used to solve for more complicated questions in reality. See there are two signal files came from the same source, and are offset by some time. Knowing that the signal propagates at 333 m/s, with sample rate 44100 Hz. It could be accurate to find the offset time, and then get the distance between two sensors, x (figure 2.3).

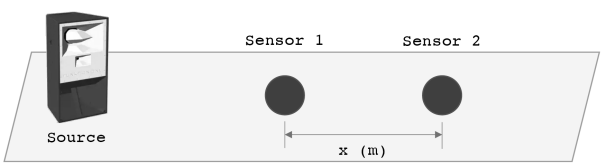


Figure 2.3: Source and sensor arrangement

After loading signals with the sample rate, it is intuitive to get the time vector and plot the signals (figure 2.4). We can observe that signals have the same shape with a time offset.

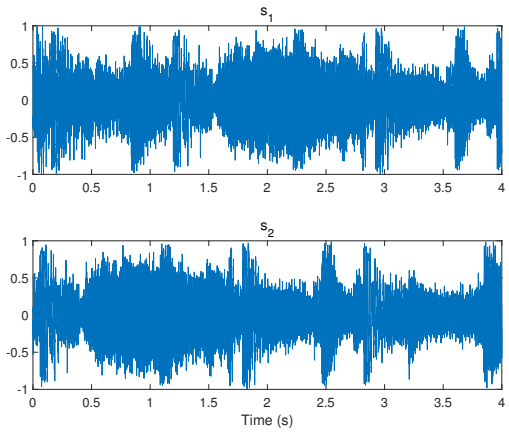


Figure 2.4: signals in time series

As shown in figure (2.5), with the x-axis (index) useless here, using the previous process in section 2.1 would be difficult to solve the problem.

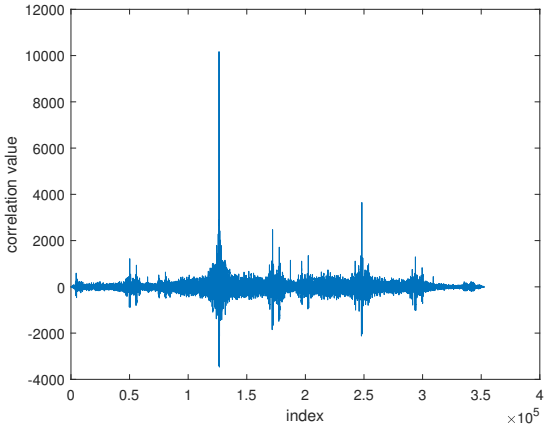


Figure 2.5: Original correlation value without lag vector

The tricky part here is that the cross-correlation vector function becomes complicated for much data in reality, which needs a simplified structure using for loop. Besides, the correlation value returned by *xcorr* function does not have the first and last zero in our implementation, as they are useless information in fact. The most important thing is it could return a lag vector for finding the offset. It is easy to get the lag vector as it should be same length as the correlation vector, with zero point at the center, which means it has same length of negative and positive values at two sides. So the improved steps involved:

1. Load two files to make them vectors.
2. Make zero vectors in the same size
3. Implement the cross-correlation vector function similar to section 2.1. Compute the cross-correlation vector.
4. Trim the first and last element (0) of the cross-correlation vector.
5. Generate the lag vector, the cross-correlation of the two measurements is maximum at a lag equal to the delay.
6. Find the maximum index of the cross-correlation vector, and find it in the lag vector, which is calculated to get offset time.
7. Computer code see appendix A.2.

The new figure of cross-correlation on lag vector becomes:

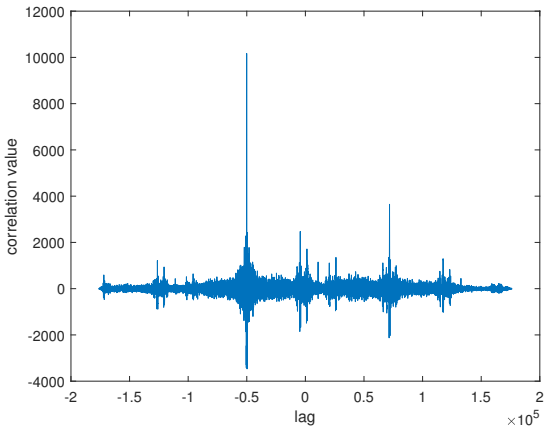


Figure 2.6: Correlation value on lag vector

The x-axis becomes lag vector here, as the lag with maximum cross-correlation obtained is -50082, the results could be computed as below:

$$timeDiff = lagDiff/Fs$$
$$distance = timeDiff \times Speed$$

So the offset time becomes -1.1356s, and the distance is -378.1702m.

2.3 Correlation using FFTs

A fast Fourier transform (FFT) is an algorithm that samples a signal over a period of time (or space) and divides it into its frequency components [2]. These components are single sinusoidal oscillations at distinct frequencies each with their own amplitude and phase. Fast Fourier transforms are widely used for many applications in engineering, science, and mathematics.

In MATLAB, FFT is a very good function for spectral analysis because of its fast and convenient features, and it could also be used to implement cross-correlation here, compared to summation of products in previous sections. Cross-correlation can be achived by completing a Fourier transform, multiplying signals, and doing an inverse Fourier transform, that could be calculated as below:

$$u \star v(\gamma) = F^{-1}\left\{(F\{u\})^* \cdot F\{v\}\right\}$$

Where the * refers to the complex conjugate. The process involves:

1. Use *fft* getting n-point discrete Fourier transform (DFT) of two vectors, that n is sum of lengths minus 1.
2. Use *ifft* to compute the inverse fast Fourier transform of DFT of the first vector *times* the complex conjugate of the second DFT.
3. Use *fftshift* to Shift zero-frequency component of the result to center of spectrum, getting the cross-correlation, r.
4. Other steps are similar to the previous process in section 2.2.
5. Computer code see appendix A.3.

The algorithm can be tested using the same problem in Section 2.2. Load and re-analyse the signals, plot the cross-correlation and get the results of offset. We can find that the new figure of cross-correlation by FFT has the same maximum value with index (positive) as figure 2.6.

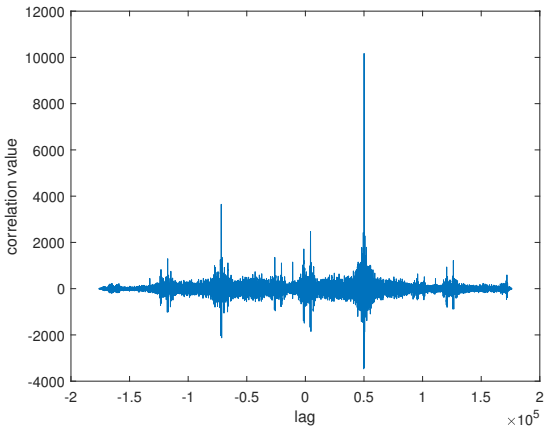


Figure 2.7: Correlation value by FFT

There is a *tic-toc* function for measuring the performance of the algorithm. As the running time of cross-correlation in section 2.2 is about 265.79 seconds, whereas it just took 0.15 seconds in section 2.3. It is obvious that correlation using FFT is much faster than the summation of a product.

2.4 Pattern Finder

Cross-correlation is not only used find signal offset, but more commonly used to find pattern matching in many objects, like signals, pictures, etc. as we said at the beginning. It could be first applied to finding all occurrences of a particular element in a signal, and then we will turn to two-dimensional pictures in the next chapter.

Because of the fast and accurate feature of FFT implementation from section 2.3, we would like to try a pattern search finding all the drum sound in a song named 'imperial_march.wav' downloaded from Internet [1]. The process involves:

1. Load the song using *audioread* to get the vector with sample rate.
2. Listen and capture a drum sound as the template. Append zeros at the end of the shorter vector (drum template) to make it have the same size as the original vector (song).
3. Correlation using FFT to get the cross-correlation, r. Generate the lag vector.
4. The tricky part is first looking at value r. From figure 2.8, we could find the relative high points (split value) are beyond 15, so try to pick 16 as the threshold finding occurrences here. If the absolute value of cross-correlation is greater than 16, add it to the index vector.
5. The offset time can be found by iterating the index vector, and then divided by the sample rate. At which point, the frequency should be 1.
6. Plot the time point with frequency 1 in red asterisk in time series of the song.
7. Computer code see appendix A.4.

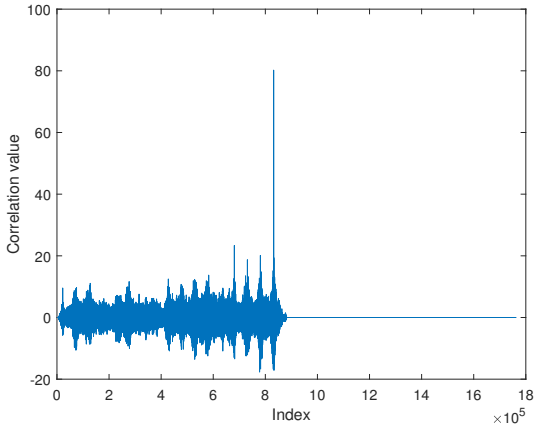


Figure 2.8: Correlation value of the song

The x-axis is time while the y-axis is frequency of the drum in figure 2.9. The results fits the drum in song well, which means cross-correlation is accurate at pattern finding in one-dimensional signal.

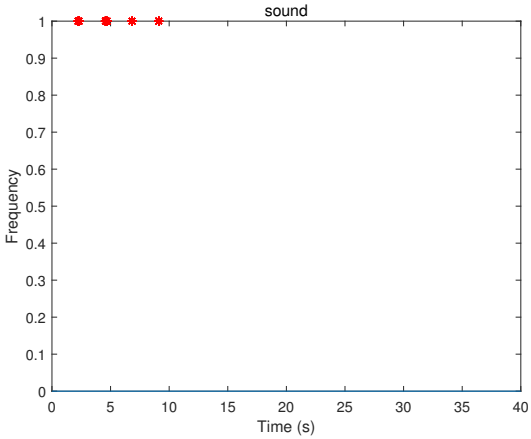


Figure 2.9: Occurrences of drum in the song

Next, we will going to expand the situation to two-dimensional matrices.

Chapter 3

Cross Correlation in two dimensions

To achieve the stereo vision system, normalised cross correlation in 2D is the most important part for finding information in pictures. There are built-in function *xcorr2* and *normxcorr2* in MATLAB, but we will first implement the function for better understanding as necessary.

3.1 Normalised Cross-Correlation in 2D

See there are two matrices, t(template) and S (search region), matrix S will always be larger than matrix t. The idea is using two nested for-loops to "lag" t over S, computing for each "lag" the cross-correlation, r. Assume that matrix t has dimensions (Mt, Nt) and matrix S has dimensions (MS, NS). When the block calculates the full output size, the equation for the two-dimensional discrete cross-correlation is:

$$C(i,j) = \sum_{m=0}^{(MS-1)} \sum_{n=0}^{(NS-1)} S(m,n) \cdot conj\left(t(m+i, n+j)\right)$$
$$where\ 0 \leq i < MS + Mt - 1, 0 \leq j < NS + Nt - 1$$

There are some points needed attention. Making use of the images rather than creating matrices manually is more effective. Besides, there are 3 sets of values representing RGB respectively for each pixel in images, it needs to get the mean value of three colors, which called *greyscale* that can get 3 sets into 1. The last thing is dealing with out-of-bounds errors with images, that we need to first create a bigger search region, that appends same size 0 as template on two sides of the original search region, and then put it at the center of the new matrix. With the example of matrix 3.1.1, where m equals to addition of the original width of the search region and 2 times of the template's width, while n is same in length.

$$\boldsymbol{c} = \begin{pmatrix} c_{11} & c_{12} & c_{13} & \dots & c_{1n} \\ c_{21} & c_{22} & c_{23} & \dots & c_{2n} \\ c_{31} & c_{32} & c_{33} & \dots & c_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & c_{m3} & \dots & c_{mn} \end{pmatrix}. \tag{3.1.1}$$

The process employed to complete the analysis involved:

1. Images are just a matrix of pixel values, so we picked two simple images instead of generating matrices.
2. Load two images using *imread*, convert RGB images to greyscale.
3. subtract the mean value so that there are toughly equal numbers of negative and positive values.
4. Make a zero matrix and put the search region in the center, getting a new search region matrix.
5. Initialize the normalised cross-correlation matrix with size of the sum of two matrices.
6. Use nested for-loops to "lag" the template over the search region, each time get the corresponding region from the new search region in step 4.
7. Compute for each "lag" the cross-correlation, r. The equation is using complex conjugate of the template.
8. Compute the normalised cross-correlation by dividing each r with square root of product of two sum of dot product.

9. Trim the useless values (NaN) surrounding the matrix.
10. Computer code see appendix A.5.

3.2 Finding the Rocket Man

Now we can make use of the cross-correlation function in two-dimensional situations to find an object from a picture, which is very similar to pattern finding drum sounds in the song. See, there is a rocket man captured from an image of maze (figure 3.2) as the template here (figure 3.1).

Figure 3.1: Rocket Man (template)

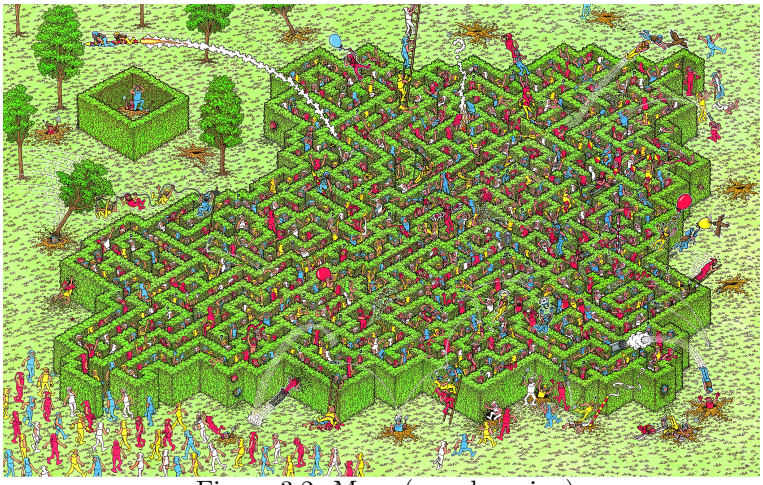


Figure 3.2: Maze (search region)

So the algorithm should involve:

1. Load two images using *imread*, convert RGB images to greyscale.
2. Use the same process as section 3.1.
3. Making the 2D cross-correlation to 1D, that could easily get the coordinate of the maximum value. Draw a red circle on the maximum value, see figure 3.3.
4. The maximum of the cross-correlation corresponds to the estimated location of the lower-right corner of the section. Use *ind2sub* to convert the one-dimensional location of the maximum to two-dimensional coordinate.
5. Draw the original image and put a red star in the center of the found section on it, see figure 3.4.
6. The run time is about 191.10 seconds, compared to *xcorr2* with just 6.08 seconds.
7. Computer code see appendix A.6.

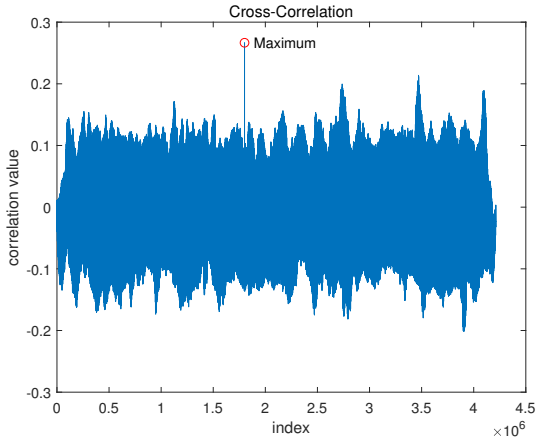


Figure 3.3: Cross Correlation with the maximum value

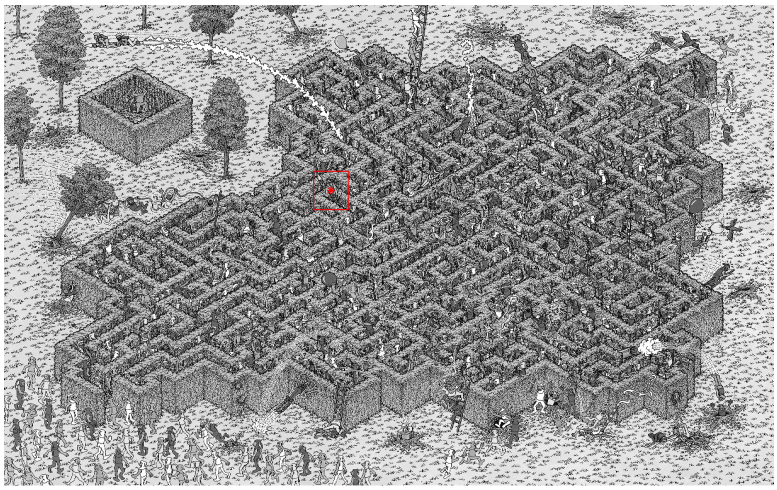


Figure 3.4: Rocket Man found in the Maze

Now we have implemented cross-correlation in both 1D and 2D, with an almost complete understanding. Next, we are going to extend the algorithm as well as moving towards a 3D Stereo Vision system.

Chapter 4

3D Stereo Vision

After finishing the important preparation with knowledge of cross-correlation, we are ready to start the Stereo Vision System. The brief work consists of fitting 'Gaussian Peak', windows comparison and creating a calibration model, etc. As we have already understood the implementation of cross-correlation well, we are going to use the built-in function *normxcorr2* directly in the following work, which is much faster and more convenient to make the project with high efficiency, but should be aware of the edge effect here, it may be necessary to remove the effect value that is artificially high due to errors in the cross-correlation. Besides, there is a case that when values of template are all the same, *normxcorr2* cannot work because of meaningless template in this case, that could be solved by adjusting parameters like template size to include different information, or annotating *checkIfFlat(T)* of the built-in *normxcorr2*, but it is not suggested by MATLAB. Another way is using the normalised cross-correlation function implemented in section 3.1, which is not suggested too because of the extremely long execution time in calibration and comparison process.

A common method for creating a stereo vision system is to use a calibration plate [3]. Thanks for the University of Melbourne, that provided 6 pairs of calibration images. The images are from a left and right camera viewing the same calibration plate at a stereo angle of approximately +/-9 °. The calibration target has white dots spaced by 50 mm in the x (horizontal) and y (vertical) directions. The calibration target is shifted to various z locations, starting at a distance of 2000 mm from the camera, and shifted in 20 mm increments towards the camera (in the negative z direction). The calibration file names contain the name of the stereo camera (i.e left or right) and the z location of the calibration target (i.e 2000), see figure 4.2.

Each of the identified dots has a known (x, y, z) in real space (in mm). The datum is like in figure left_2000.tiff, with x=0mm , y=0mm located at the 11th dot from the left, in the lowest row of dots. The coordinates of each common dot from the left image (il,jl) and the right image (ir,jr) are uniquely associated with a given (x,y,z) in real space. For example the coordinates of the lowest left-hand dot in the left and right images shown in figure left_2000.tiff correspond to the real space coordinates (-500, 0, 2000), thus we could get all coordinates of dots in real space (figure 4.1). So the next step needed to do is finding all dots in pixel coordinates.

(-500,800)	(-450,800)	(-400,800)	(-350,800)	(-300,800)	(-250,800)	(-200,800)	(-150,800)	(-100,800)	(-50,800)	(0,800)	(50,800)	(100,800)	(150,800)	(200,800)	(250,800)	(300,800)	(350,800)	(400,800)	(450,800)	(500,800)
(-500,750)	(-450,750)	(-400,750)	(-350,750)	(-300,750)	(-250,750)	(-200,750)	(-150,750)	(-100,750)	(-50,750)	(0,750)	(50,750)	(100,750)	(150,750)	(200,750)	(250,750)	(300,750)	(350,750)	(400,750)	(450,750)	(500,750)
(-500,700)	(-450,700)	(-400,700)	(-350,700)	(-300,700)	(-250,700)	(-200,700)	(-150,700)	(-100,700)	(-50,700)	(0,700)	(50,700)	(100,700)	(150,700)	(200,700)	(250,700)	(300,700)	(350,700)	(400,700)	(450,700)	(500,700)
(-500,650)	(-450,650)	(-400,650)	(-350,650)	(-300,650)	(-250,650)	(-200,650)	(-150,650)	(-100,650)	(-50,650)	(0,650)	(50,650)	(100,650)	(150,650)	(200,650)	(250,650)	(300,650)	(350,650)	(400,650)	(450,650)	(500,650)
(-500,600)	(-450,600)	(-400,600)	(-350,600)	(-300,600)	(-250,600)	(-200,600)	(-150,600)	(-100,600)	(-50,600)	(0,600)	(50,600)	(100,600)	(150,600)	(200,600)	(250,600)	(300,600)	(350,600)	(400,600)	(450,600)	(500,600)
(-500,550)	(-450,550)	(-400,550)	(-350,550)	(-300,550)	(-250,550)	(-200,550)	(-150,550)	(-100,550)	(-50,550)	(0,550)	(50,550)	(100,550)	(150,550)	(200,550)	(250,550)	(300,550)	(350,550)	(400,550)	(450,550)	(500,550)
(-500,500)	(-450,500)	(-400,500)	(-350,500)	(-300,500)	(-250,500)	(-200,500)	(-150,500)	(-100,500)	(-50,500)	(0,500)	(50,500)	(100,500)	(150,500)	(200,500)	(250,500)	(300,500)	(350,500)	(400,500)	(450,500)	(500,500)
(-500,450)	(-450,450)	(-400,450)	(-350,450)	(-300,450)	(-250,450)	(-200,450)	(-150,450)	(-100,450)	(-50,450)	(0,450)	(50,450)	(100,450)	(150,450)	(200,450)	(250,450)	(300,450)	(350,450)	(400,450)	(450,450)	(500,450)
(-500,400)	(-450,400)	(-400,400)	(-350,400)	(-300,400)	(-250,400)	(-200,400)	(-150,400)	(-100,400)	(-50,400)	(0,400)	(50,400)	(100,400)	(150,400)	(200,400)	(250,400)	(300,400)	(350,400)	(400,400)	(450,400)	(500,400)
(-500,350)	(-450,350)	(-400,350)	(-350,350)	(-300,350)	(-250,350)	(-200,350)	(-150,350)	(-100,350)	(-50,350)	(0,350)	(50,350)	(100,350)	(150,350)	(200,350)	(250,350)	(300,350)	(350,350)	(400,350)	(450,350)	(500,350)
(-500,300)	(-450,300)	(-400,300)	(-350,300)	(-300,300)	(-250,300)	(-200,300)	(-150,300)	(-100,300)	(-50,300)	(0,300)	(50,300)	(100,300)	(150,300)	(200,300)	(250,300)	(300,300)	(350,300)	(400,300)	(450,300)	(500,300)
(-500,250)	(-450,250)	(-400,250)	(-350,250)	(-300,250)	(-250,250)	(-200,250)	(-150,250)	(-100,250)	(-50,250)	(0,250)	(50,250)	(100,250)	(150,250)	(200,250)	(250,250)	(300,250)	(350,250)	(400,250)	(450,250)	(500,250)
(-500,200)	(-450,200)	(-400,200)	(-350,200)	(-300,200)	(-250,200)	(-200,200)	(-150,200)	(-100,200)	(-50,200)	(0,200)	(50,200)	(100,200)	(150,200)	(200,200)	(250,200)	(300,200)	(350,200)	(400,200)	(450,200)	(500,200)
(-500,150)	(-450,150)	(-400,150)	(-350,150)	(-300,150)	(-250,150)	(-200,150)	(-150,150)	(-100,150)	(-50,150)	(0,150)	(50,150)	(100,150)	(150,150)	(200,150)	(250,150)	(300,150)	(350,150)	(400,150)	(450,150)	(500,150)
(-500,100)	(-450,100)	(-400,100)	(-350,100)	(-300,100)	(-250,100)	(-200,100)	(-150,100)	(-100,100)	(-50,100)	(0,100)	(50,100)	(100,100)	(150,100)	(200,100)	(250,100)	(300,100)	(350,100)	(400,100)	(450,100)	(500,100)
(-500,50)	(-450,50)	(-400,50)	(-350,50)	(-300,50)	(-250,50)	(-200,50)	(-150,50)	(-100,50)	(-50,50)	(0,50)	(50,50)	(100,50)	(150,50)	(200,50)	(250,50)	(300,50)	(350,50)	(400,50)	(450,50)	(500,50)
(-500,0)	(-450,0)	(-400,0)	(-350,0)	(-300,0)	(-250,0)	(-200,0)	(-150,0)	(-100,0)	(-50,0)	(0,0)	(50,0)	(100,0)	(150,0)	(200,0)	(250,0)	(300,0)	(350,0)	(400,0)	(450,0)	(500,0)

Figure 4.1: Dots' coordinates in real space

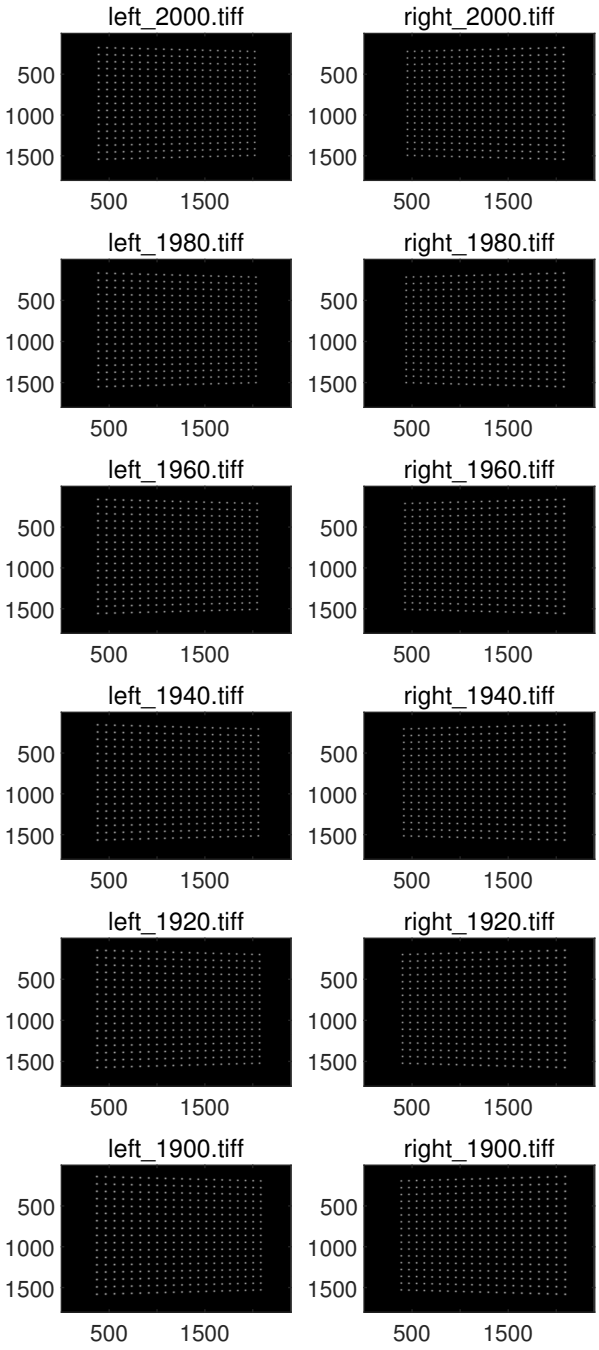


Figure 4.2: Calibration images

4.1 Dot Detection with Gaussian template

The computer vision system then needs to determine where the dots on the plate are. One of the most reliable methods is the Gaussian Peak detection method. In this approach, a 2D Gaussian object is used as a template and passed over a search region to identify peaks, in our case dots or bright spots, the peak of the cross-correlation should be the center of dots.

We could first find and plot the locations of the dots from left_2000.tiff on an equal axis to test the idea, the

process is almost identical to searching for the Rocket Man in section 3.2, which involves:

1. Load the image using *imread*, convert RGB images to greyscale.
2. Generate the Gaussian template (shown in figure 4.3 and 4.4).
3. Use *normxcorr2* to traverse template on the search region.
4. With the observation of the image, it is easy to find that there are 17 rows with 21 columns of dots. Find the coordinates of the maximum cross-correlation using 17*21 times for-loop. But note that here is a problem with repetition finds for the same dot even though we delete the maximum value after each finding, as the second greatest value might be within the area of the same dot, so we need to deal with the area of the maximum value found each time.
5. Because the maximum value is the center point of the dot, we assume that the area of Gaussian template around the value are relative high points which could affect the next find. A good way to do is making the area to zero after finding the maximum value each time.
6. Plot red dots on the original image with equal axis (figure 4.5).
7. Computer code see appendix A.7.

0	5	11	5	0
5	53	117	53	5
11	117	255	117	11
5	53	117	53	5
0	5	11	5	0

Figure 4.3: 5*5 Gaussian Matrix

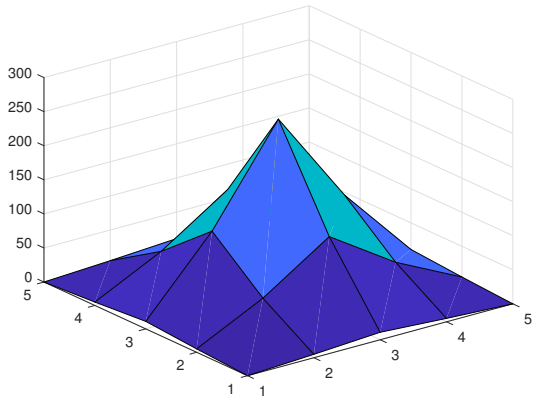


Figure 4.4: 2D Gaussian

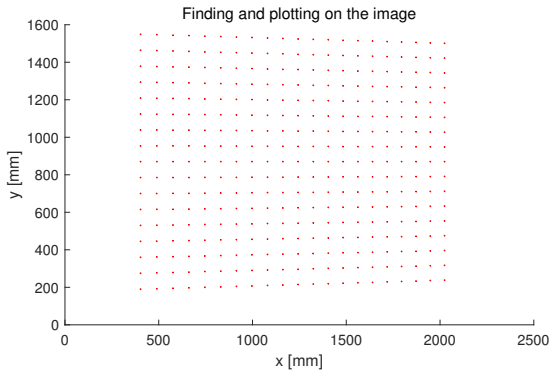


Figure 4.5: Dots on an equal axis

From figure 4.5, the result seems very accurate as in left_2000.tiff, that the algorithm is effective enough to use, finding all pairs of images and creating a calibration model for 3D stereo vision system.

4.2 Creating Calibration Model

Figure 4.1 has shown all coordinates of (x,y) in real space of dots, while the z coordinate is changing from 2000 to 1900. Our next step is to create a program which imports all the calibration images and records the pixel and real locations of all of the dots in each image. This will require a fitting tool to create a 4D surface fit that could connect all pixel space to real space within your calibrated zone.

The target is to calibrate the model with polyfit algorithm, which is effective and particular useful for barrel distortion from lens. Here is a function named *polyfitn* from MATLAB file exchange that fits a general polynomial regression model in n dimensions, it allows users to create models with more than one independent variables. The calibration model fit is a four-dimensonal surface fit with the following form:

$$X = fx\,(il, jl, ir, jr)$$
$$Y = fy\,(il, jl, ir, jr)$$
$$Z = fz\,(il, jl, ir, jr)$$

The process involves:

1. Make 7 groups of zero matrices, initialising [il,jl,ir,jr,Xre,Yre,Zre]=deal(zeros(17*21*6,1)), where il, jl, ir, jr refer to coordinates of dots in left and right images, and Xre, Yre and Zre mean real coordinates here.
2. Put 6 pairs of real coordinates of dots to Xre, Yre and Zre respectively, the Xre and Yre are the same as figure 4.1 among 6 pairs, while the Zre coordinates in each pair are all the same like 2000, 1980, ..., till 1900.
3. As all dots' coordinates could be found with the algorithm in section 4.1, but the point is how to correspond each dot to its location in the correct order? There is parallax distortion because of camera axis not being orthogonal to the plate.
4. From figure 4.2 we can see that the i coordinates of dots are same in each column whereas the j coordinates are different in each row. The base line is 9th row of the plates, which is the horizontal line. In the left image the dots above that row are in descending order, while the dots below that row are in ascending order. The right images are converse.
5. Use *sort* function to sort each row to get the right order of dots, that could get all coordinates of il, jl, ir, jr.
6. Finally use *polyfitn* to fit the model as the first demo code below.
7. As we got the calibration model now, we could input a group of [il,jl,ij,jr] to test the accuracy of model, as the second demo code. The result is pretty good as the error is less than +/-0.5 mm.
8. Complete code see appendix A.8.

```
% polyfit for X
x_input = [il , jl , ir , jr] ;
depvar_x = X(:);
x_fit = polyfitn(x_input, depvar_x, 3);
sx = polyn2sym(x_fit);

% polyfit for Y
y_input = [il , jl , ir , jr] ;
depvar_y = Y(:);
y_fit = polyfitn(y_input, depvar_y, 3);
sy = polyn2sym(y_fit);

% polyfit for Z
z_input = [il , jl , ir , jr] ;
depvar_z = Z(:);
z_fit = polyfitn(z_input, depvar_z, 3);
sz = polyn2sym(z_fit);
```

```
%% test for the pair of Z=2000mm images
%Get the calibration model
```

```
X = matlabFunction(sx);
Y = matlabFunction(sy);
Z = matlabFunction(sz);

Xreal_model = X(i_left_2000,j_left_2000,i_right_2000,j_right_2000);
Yreal_model = Y(i_left_2000,j_left_2000,i_right_2000,j_right_2000);
Zreal_model = Z(i_left_2000,j_left_2000,i_right_2000,j_right_2000);
```

Next, we can make use of the calibration model on a pair of images from resource test_left_1.tiff and test_right_1.tiff (fig 4.6) without real space information, to create a 3D reconstruction of the image pairs and display the results in figure 4.7. The basic idea is similar to finding dots as before, which could get [il,jl,ij,jr] of each dot in pixel, and then input them to the calibration model. The computer code including plotting 3D image refers to A.9.

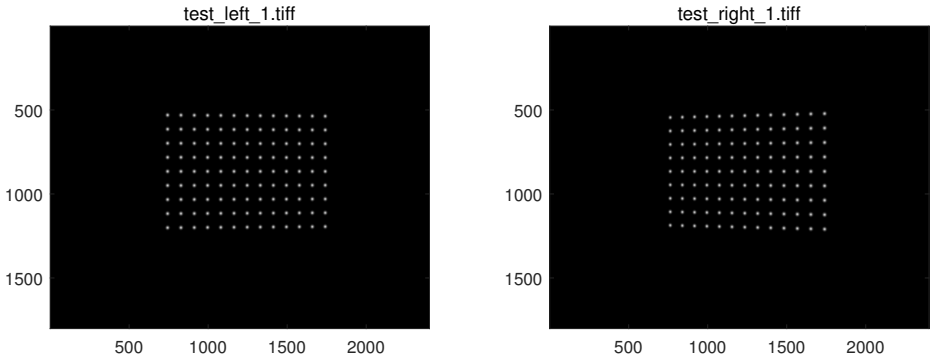


Figure 4.6: The first test pair of images

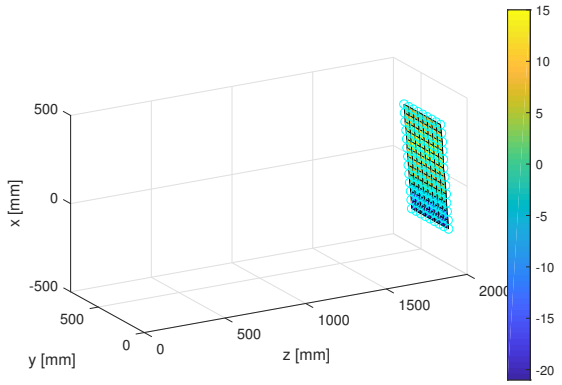


Figure 4.7: Real coordinates of figure 4.6

The gradient color in figure 4.7 means difference of i coordinate of the same dot (dx), which is more important here, because a big dx (big disparity between the found location in each camera) might mean the object is close to the cameras, leading to the further implementation.

4.3 Image Comparison

As the truth in our life, the idea is easy to prove. Hold your finger up at arms length away in front of your face. Close one eye and observe the location of your finger, and then close the other eye. Note the location with no obvious change, which means dx is small. Now hold your finger just in front of your nose, do the same process, you will find a big change in location. How far away is what the calibration images are for. But are dx and dy both useful or which one is more important? Is there any limitation with the program? The answers will be discussed later.

In the previous sections, the model is only used in simple situations as the dots are easy to find using 'Gaussian peak' template, but the cross-correlation is also a very powerful tool which can sensitively and accurately identify patterns in complex images. As humans, most of us can easily recognise two stereo images are of the same scene,

only taken from different angles or positions, but how does the computer know? There comes a program which compares two complex stereo images from Internet, using window cross-correlation, identifying the scene and getting (dx,dy) of each window.

The program involves:

1. Load a pair of stereo images using *imread*, convert RGB images to greyscale.
2. Set a window size in advance like 64 pixel means a 64*64 square window of the image.
3. Break up the left image into windows, a problem here is that the length and width of image might not be divisible by window size, that we need to pad zeros to the right side as well as the bottom side of the image, as the scan will start at the top-left window. These windows are considered as templates in the future scan.
4. Make corresponding windows on the right image, which are search regions to find similar features of the templates. The default search region is consider 3 times larger than the template, which means the window size in right image is 64*3 pixel. The tricky way to correspond the window is first padding zeros to the right image in the same way as left image, and then padding extra zeros windows (in window size) surrounding it, which means top-left window is compared to top-left window, the second left is compared to the second and so on. See figure 4.8 for a brief understanding, each window is a square in the same size, scan the window 1 on left with blue background around the corresponding search region on right with blue background as well. After finding coordinates of the template (windows 1) using normalised cross-correlation, dx and dy are calculated as relative offset with the window 1 in the right image.
5. Return dx and dy for the difference in pixel location for all windows.
6. For a detailed example, there are two stereo images from Internet shown in figure 4.9, that could be run in our program. The result is displayed as figure 4.10, the difference in x and y are small and gentle, which proves that the images are of the same scene but different angle. The relative high values may be because of the edge and dark areas are lack of information in the image, that need more optimisation in the future.
7. Computer code see appendix A.10.

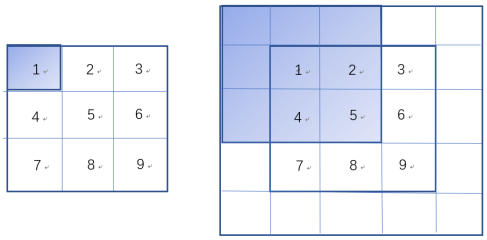


Figure 4.8: Windows cross-correlation

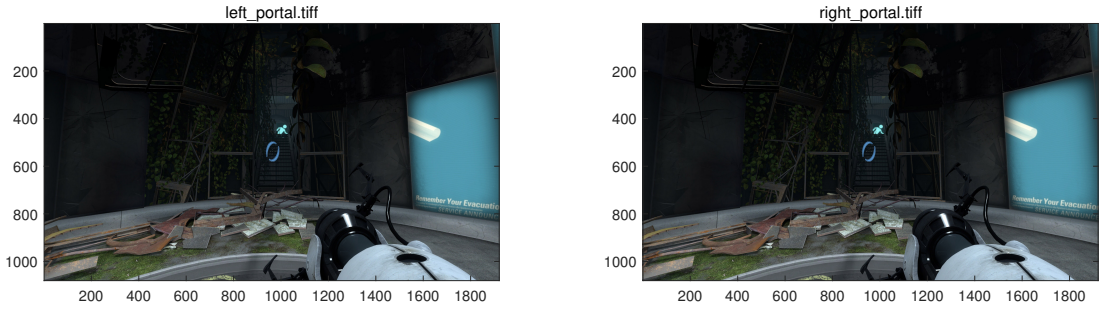


Figure 4.9: Portal stereo images

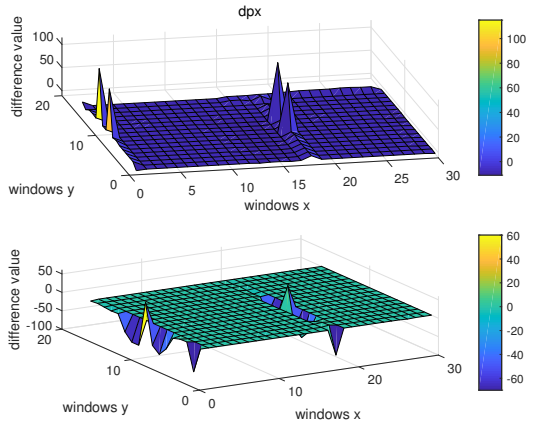


Figure 4.10: Portal stereo images' dx and dy

It is easy to get dx and dy of dots in figure 4.5, but if there is no dots (obvious information) in the images, instead there is lots of noise, like in test_left_2.riff and test_right_2.riff (figure 4.11), which is impossible to find any useful information. A valid way for dealing with these images is to use the window comparison algorithm. The results of dx and dy of the second test pair with window size 64 is shown in figure 4.12, the code is the same as appendix A.10 except loading with different pair of images.

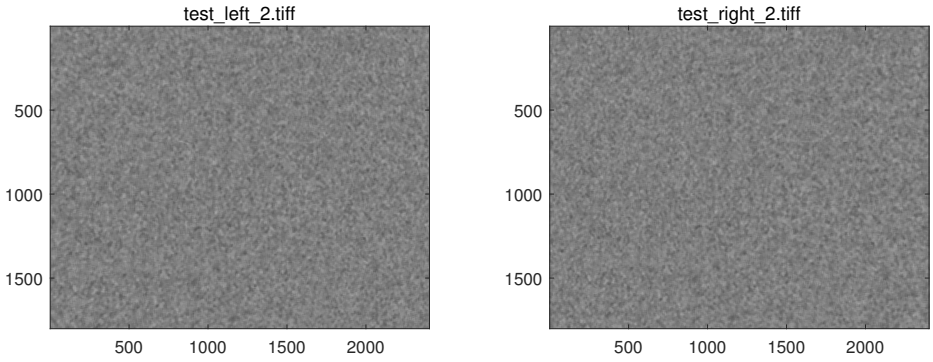


Figure 4.11: The second test pair of images

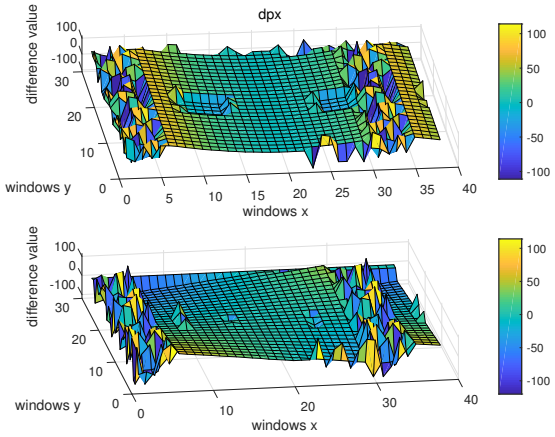


Figure 4.12: dx and dy of figure 4.11

It seems that there are many spurious vectors (high dx and dy due to errors in cross-correlation) near the uneven edges. However, we can still find very useful information as there are two depressed square area of dx on two sides, which is consistent with the two square areas in figure 4.11. It seems that the dx has more information that is more useful than dy, the explanation may be most animals have eyes in x direction, as well as the camera movement, if the stereo vision is mimic some animals that have different mechanism of eyesight, the dy might become more useful instead. As the result, we will adapt dx as the main difference in the project, which is used to reconstruct the 3D real space like the gradient color of dx shown in figure 4.7, the difference of the second pair

is that only plotting the center point of each window instead of dots, because we just know pixel coordinates of the windows. First getting the calibration model as necessary, and then plot the 3D reconstruction as the same process as the first test pair, the result is shown in figure 4.13. Computer code of 3D reconstruction is changed in appendix A.11

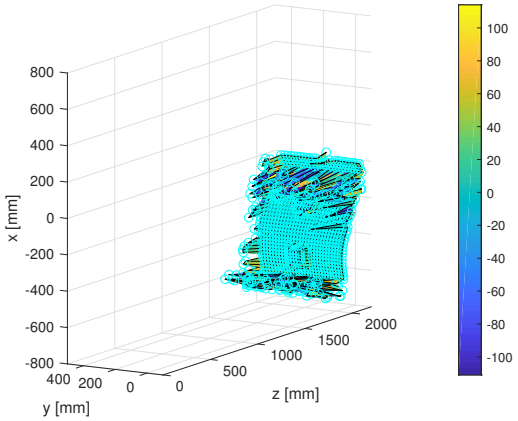


Figure 4.13: Real coordinates of figure 4.11

There are two general ways to optimise the result, one is just removing spurious vectors directly, that we could try to remove any dx and dy that bigger than 50 and just plot the surface for better observation. The results become:

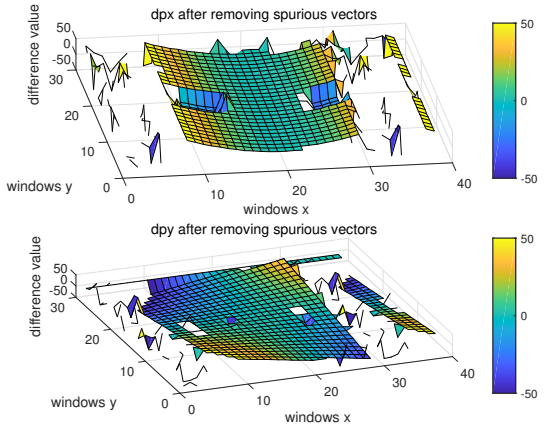


Figure 4.14: dx and dy after removing spurious vectors

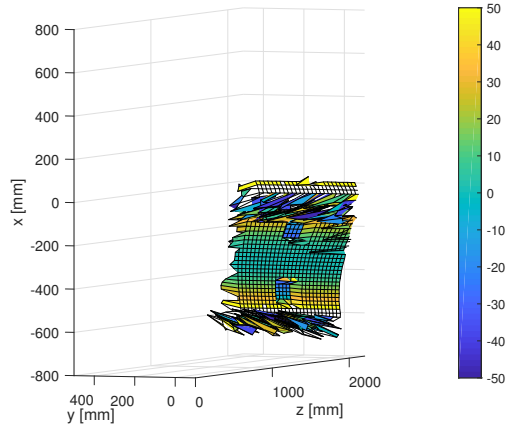


Figure 4.15: Real coordinates after removing spurious vectors

The second but more effective way is optimisation of the algorithm for better image comparison and detection, which will be implemented in the last chapter.

Chapter 5

Optimisation and Application

The idea of optimisation is tuning parameters to apply the cross-correlation, it could also be considered as trade-off between accuracy and speed, as higher accuracy needs more information as well as computation basically. There are 5 general optimisation strategies for image comparison implemented in chapter 4:

1. Window size
2. Window overlap
3. Search region
4. Multiple pass
5. Sub-pixel

After optimisiation, the algorithm should be applied into some real photos taken by our smart phones.

5.1 Image Comparison Optimisation

First we will come to Window size optimisation, the idea of this strategy is that the bigger size of the window is, the more completed information the window could carry. As the same example with figure 4.9 and figure 4.11, as the window size is increased to 100, the parameter could be changed in code A.10 directly, and the results improved as figure 5.1 and 5.2.

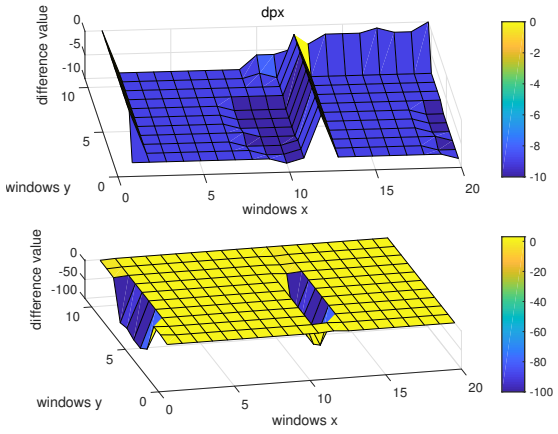


Figure 5.1: Window size optimisation of figure 4.9

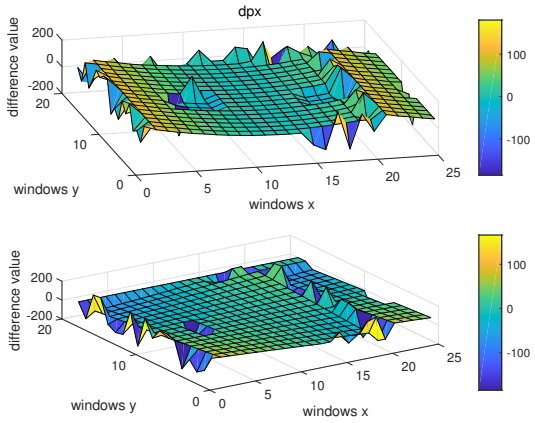


Figure 5.2: Window size optimisation of figure 4.11

The program for figure 4.9 took about 8 seconds with window size 64, but costed 12 seconds with window size 100. The program becomes slower as the computation of cross-correlation is more complicated with increasing window size. However, the accuracy is better since the error becomes small obviously in figure 5.1, even detected some part of no difference (lens part in the image), though still with some error. As in figure 5.2 compared to figure 4.12, the spurious vectors become much more smooth on two sides.

The following 3 strategies are more complex because the template and search region need to be redistributed and computed, but the basic ideas are simple as follows:

1. Window overlap could overlap the part of neighboring windows, sometimes the window will lose some important information as it separate the consistent useful information, but overlap could get rid of the problem as shown in figure 5.3.

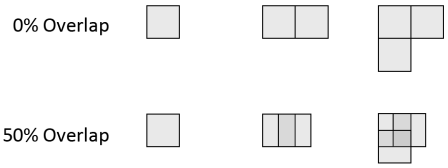


Figure 5.3: Example of window overlap

2. The idea of increasing search region is similar to increasing window size as well as window overlap, they three all could get more information in the search region part, which could be more accurate to find and identify the objects in the left image (template) on the right image (search region). The demo is like below.

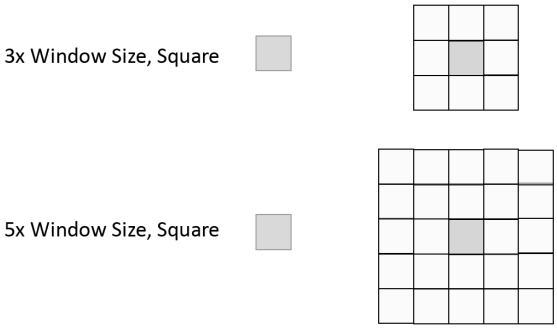


Figure 5.4: Example of search region

3. The most complicated implementation is multi-pass, as it runs more than one rounds of image comparison, which will cost much computation but necessary in some cases, figure 5.5 shows a second pass example. The left blue square containing 1,2,3,4 is a window with size 64, it will be found in right blue square with a brief location but not accurate as the detailed location of 1,2,3,4 are different. The first pass returns a pair (dx_est, dy_est)

used as an estimate for the second pass. In the second pass, the search region in the right image is centred at the location of the template plus (dx_est, dy_est), so it is different in breaking the corresponding windows in the right image as before, details of computation are complex which could be referred in code.

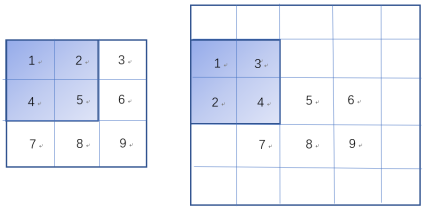


Figure 5.5: Example of second pass

These three optimisations are included in one program separately to make a better comparison. The default parameters are 64 pixel window size with 0% overlap, 3 times window size of search region and one pass as in section 4.3, whereas the optimisation parameters are 64 pixel window size, with 50% overlap, 5 times window size of search region and two passes. As the results of figure 4.9, the performance of bigger search region (figure 5.7) seems the best one but took the longest time among three. The second pass implementation looks a little messy might because the 32 pixel window size is too small to carry enough information, as we increased the size to 128 pixel it becomes more flat, and the speed performance is the best among three. It seems that 50% overlap has no obvious improvement in this case, and the time cost is almost close to the search region optimisation. Computer code refers to appendix A.12, though there is still a self implemented normalised cross-correlation function contained at the end, it is not suggested to use because of the frustrating performance in speed.

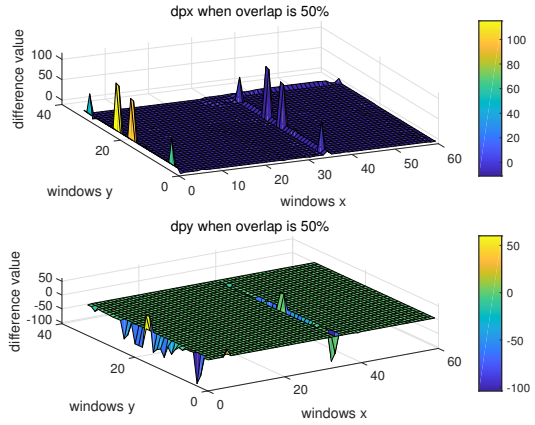


Figure 5.6: Window overlap optimisation of figure 4.9

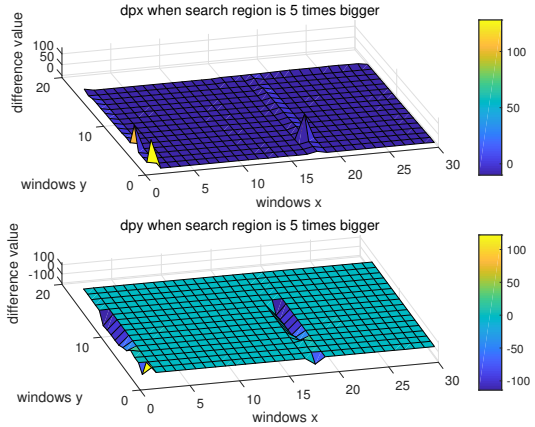


Figure 5.7: Search region optimisation of figure 4.9

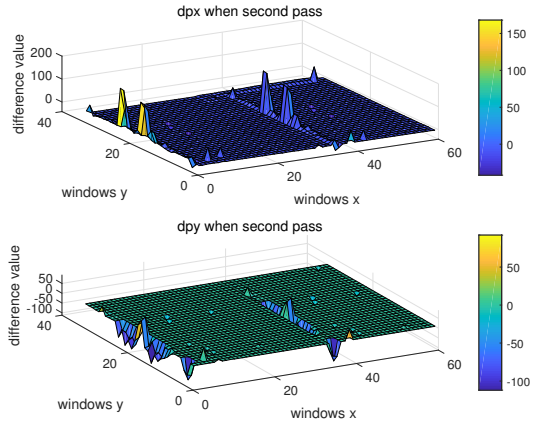


Figure 5.8: Second pass optimisation of figure 4.9

Now the best optimisation (bigger search region) could be used in figure 4.11, the result is optimised as:

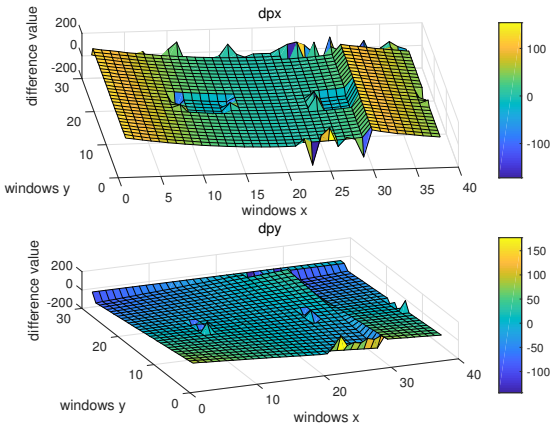


Figure 5.9: Search region optimisation of figure 4.11

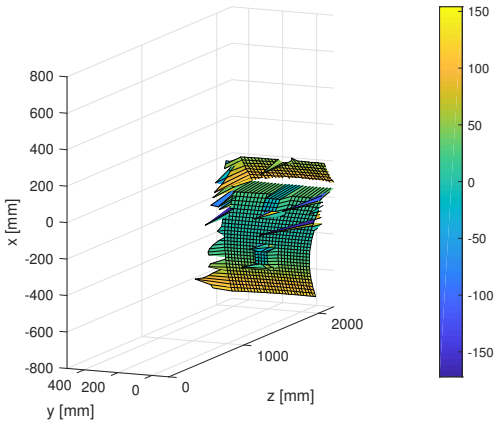


Figure 5.10: Optimised real location of figure 4.11

As for sub-pixel optimisation, it is a very good and sensitive strategy for identifying very tiny object or comparing small images. The benefit is magnifying pixel of objects with the image, which could be implemented by *interp2*. The function could return interpolated values of original pixel points with spacing of 0.1 using linear interpolation. We tested with the images ten times smaller of figure 4.9, the result is quite okay but not shown for concise report. The code can be referred to appendix A.13.

There is a last pair of test images in figure 5.11, which is very difficult to distinguish with the second test pair (figure 4.11) with humans' eyes. However, it is convenient and effective to identify using the optimised comparison

algorithm, as shown in figure5.12 and 5.13. The result is clear that there is just one depressed square area at the center, which is different from figure 4.11. The computer code is identical to appendix A.11 except loading a different pair of images.

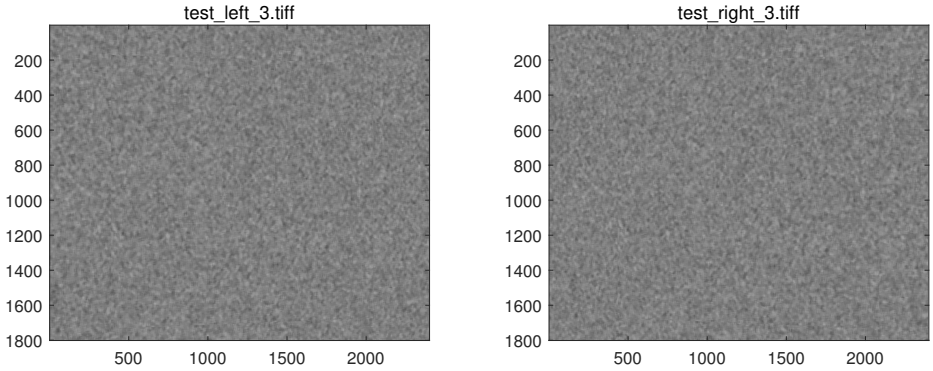


Figure 5.11: The third pair of test images

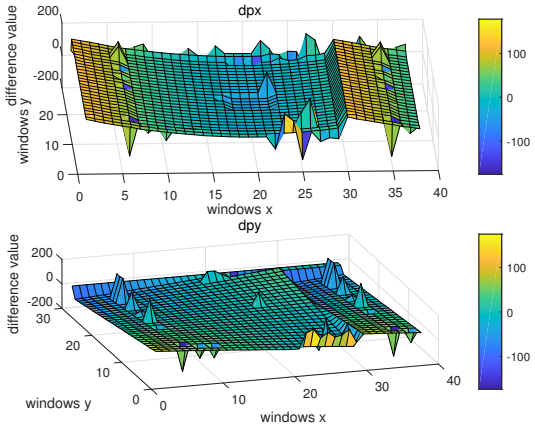


Figure 5.12: Search region optimisation of figure 5.11

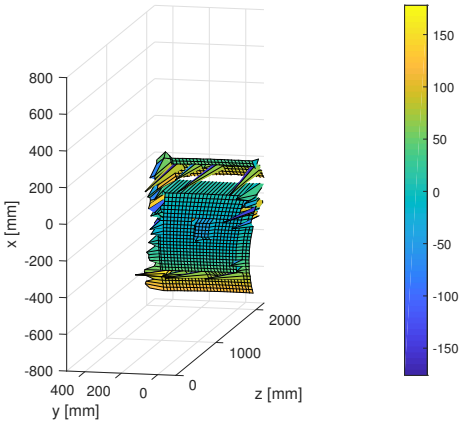


Figure 5.13: Optimised real location of figure 5.11

5.2 Application

The previous images are all from resources which might be calibrated in advance that fit comparison well. There is an Android app named 3DSteroid on Google play Store, which could take stereo photos with a datum line. Figure 5.14 is a pair of stereo images taken by an Android phone with the app. As the resolution of images is very high, we could double the window size to 128 for ideal size, with the optimised search region. The result is shown in figure 5.15, the bottle is successfully found at the middle of dx. It seems that the algorithm is valid to

applied to real objects or situations, although with some errors and limitations, that will be discussed in the last section.

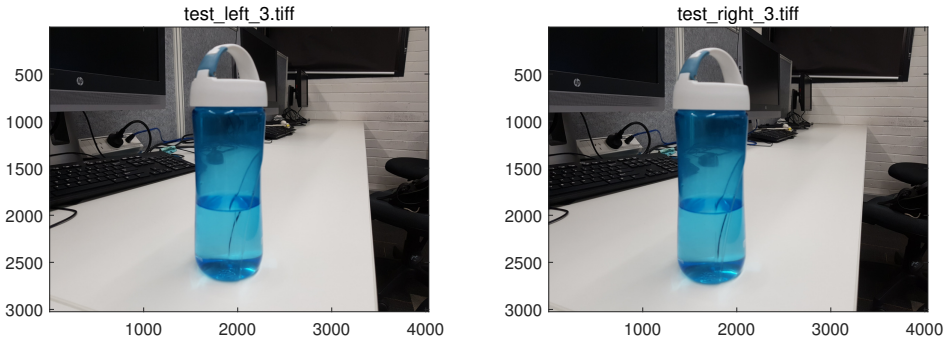


Figure 5.14: Bottle in the computer lab

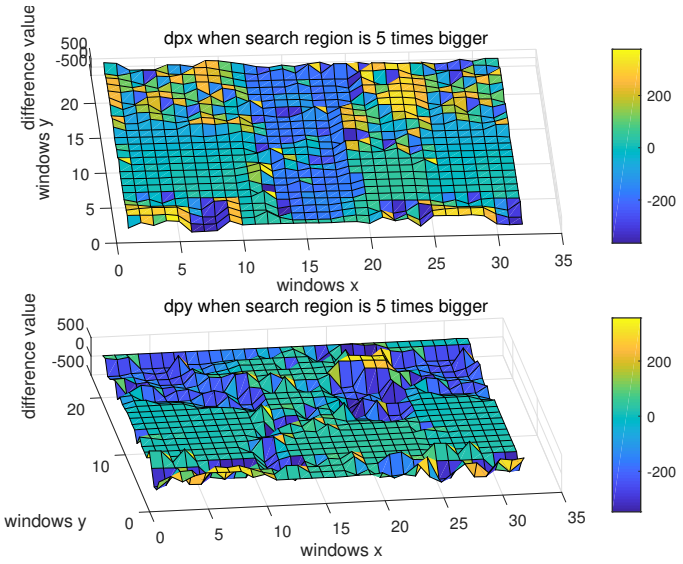


Figure 5.15: Application of figure 5.14

5.3 Conclusion

Overlap and multi-pass optimisations have no evident advantages, because the window size is small for the whole image, making the information in each window is limited. Useful information (objects) are usually bigger than the window, so whatever 0% overlap or 50% overlap has no difference in containing enough useful information, unless the image contains lots of objects that each window might contain one object like a word or a symbol. 50% overlap could be helpful to include all complete information without segmentation, and the explanation can also make sense on most multi-pass situations. Increasing the search region is an effective and convenient way for finding the template even though it shifted far from the original location, which is easy to understand.

From the project, after implementing a series of algorithms as well as figuring out the challenges of complicated computation, we got a good understanding of basic idea and structure of Computer Stereo Vision. There are two ways calculating cross-correlation, including summation of a product and FFT, both are very useful in offset detecting and template finding, while 'Gaussian peak' is a helpful method for determining where the dots on the calibration plate for the computer vision system. Corresponding and transformation (polynomial surfaces) are effective in camera calibration, and window comparison with optimisation is a general way to deal with different cases of the stereo photos. All algorithms work together achieving the 3D Stereo Vision system for extraction of useful 3D information from digital images. Though there are some limitations of the system, that our calibration model is limited that could only find the dot objects, it is hard to find the depth of a detailed real object, like a pen or cup in the photos. It needs more complicated calibration sets as well as smart algorithms that can detect the object we want, which could be extended in the future.

Appendix A

Computer code

A.1 Cross-Correlation in 1d

```
% 1. Spatial Cross Correlation + Normalised Spatial Cross Correlation in 1d

% 1) See A and B are random vectors with the same size
A = rand(1,3);   % generate vector A using rand()
B = rand(size(A));% generate vector B with the same size

ZEROS = zeros(size(A));
% Put same size of zero at the beginning and end of B.
B = horzcat(ZEROS,B,ZEROS);

% Iteration time of the for loop
times = length(B)-length(A)+1;
% Initialize the correlation vector r
r = zeros(1,times);

for i = 1:times
    r(i) = A(1) * B(i) + A(2) * B(i+1) + A(3) * B(i+2);
end
plot(r);
xlabel('index')
ylabel('correlation value')

%2) Normalised the correlation vector

% Initialize the correlation vector r
normalised_r = zeros(1,times);
normalised_A = A/norm(A);
normalised_B = B/norm(B);
%r2 = xcorr(normalised_A, normalised_B, 'none');

for i = 1:times
    normalised_r(i) = normalised_A(1) * normalised_B(i) + normalised_A(2) * normalised_B(i+1) + ...
        normalised_A(3) * normalised_B(i+2);
end
figure
plot(normalised_r);
xlabel('index')
ylabel('correlation value')
```

A.2 Finding Signal Offset

```
% 2. Signal Offset

% read the files
text1 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment1','sensor1_data_general(1).txt');
text2 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment1','sensor2_data_general(1).txt');
```

```
s1 = transpose(dlmread(text1,'\t',1,0));
s2 = transpose(dlmread(text2,'\t',1,0));

% load sample rate, speed and create time vectors and plot the signals
Fs = 44100;
Speed = 333;
t1 = (0:length(s1)-1)/Fs;
t2 = (0:length(s2)-1)/Fs;

% plot the signals with time series
subplot(2,1,1)
plot(t1,s1)
title('s_1')
subplot(2,1,2)
plot(t2,s2)
title('s_2')
xlabel('Time (s)')

%-------------------------------------------------------------------------

% make zero vector in the same size and concat to s2
N = zeros(size(s1));
s2 = horzcat(N,s2,N);

% calculate the time of cross correlation function
tic
r = correlation_vector(s1,s2);
toc

% plot the original correlation vector using customized function
figure
plot(r);
xlabel('index')
ylabel('correlation value')

%trim the first and the last element (0)
acor = r(2:end-1);
% generate lag vector
lag = zeros(1,length(s1)*2-1);
init = (length(s1)-1)*(-1);
for i = 1:(length(s1)*2-1)
    lag(i) = init + i - 1;
end

% replot the cross correlation over lag vector
figure
plot(lag,acor);
xlabel('lag')
ylabel('correlation value')
% find the max correlation value with its index, which could used to get
% the delay
[~,I] = max(abs(acor));
lagDiff = lag(I);
timeDiff = lagDiff/Fs;
distance = timeDiff * Speed;

%% correlation function
function r = correlation_vector(x,y)
if ~isvector(x)
    error('Input must be a vector')
end
if ~isvector(y)
    error('Input must be a vector')
end
% Iteration time of the for loop
times = length(y)-length(x)+1;
% Initialize the correlation vector r
r = zeros(1,times);
for i = 1:times
    for j = 1:size(x,2)
        r(i) = r(i) + x(j) * y(j+i-1);
    end
end
end
```

A.3 Correlation using FFTs

```
% 5. Spectral Cross Correlation

%------------------------------------------------------------------------------
text1 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment1','sensor1_data_general(1).txt');
text2 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment1','sensor2_data_general(1).txt');

s1 = transpose(dlmread(text1,'\t',1,0));
s2 = transpose(dlmread(text2,'\t',1,0));

% s1 = [1 2 3 4 5];
% s2 = [6 7 8 9 10];

lengtha = length(s1);
lengthb = length(s2);
l = lengtha+lengthb-1;

tic
% correlation using FFT
r = ifft(fft(s1,l).*conj(fft(s2,l)));
r = fftshift(r);
toc

% generate lag vector
lag = zeros(1,length(s1)*2-1);
init = (length(s1)-1)*(-1);
for i = 1:(length(s1)*2-1)
    lag(i) = init + i - 1;
end

Fs = 44100;
Speed = 333;

[~,I] = max(abs(r));
lagDiff = lag(I);
timeDiff = lagDiff/Fs;
distance = timeDiff * Speed;
```

A.4 Pattern Finder

```
% 6. Bonus: Pattern Finder

% read the audio
song = fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment1','imperial_march.wav');
[y,Fs] = audioread(song);

% plot the original sound (search region)
subplot(2,1,1)
t1 = (0:length(y)-1)/Fs;
plot(t1,y)
title('original sound')
xlabel('Time (s)')

% transpose to make it a vector
y = transpose(y);

template = zeros(1,length(y));
template(1:10000) = y(50001:60000);

% plot the template sound
subplot(2,1,2)
t2 = (0:length(template)-1)/Fs;
plot(t2,template)
title('template sound')
xlabel('Time (s)')

% get the length of r
```

```
lengtha = length(template);
lengthb = length(y);
l = lengtha+lengthb-1;

% correlation using FFT
r = ifft(fft(template,l).*conj(fft(y,l)));
r = fftshift(r);

% generate lag vector
lag = zeros(1,length(y)*2+1);
init = length(y)*(-1);
for i = 1:(length(y)*2+1)
    lag(i) = init + i - 1;
end


% Use the plot to find the high points (split value) at about 15-20.
figure
plot(r)
xlabel('Index')
ylabel('Correlation value')
% initialize the index vector
I=[];
% try 16 to find the occurrences here
for i = 1:length(r)
    if (abs(r(i)) > 16)
        I = [I i];
    end
end

% Show the resulting correlation vector and then line up the occurrences of the element on a ...
    plot of the
% signal (x-axis: time, y-axis: frequency)
figure
y = zeros(1,length(y));
plot(t1,y)
title('sound')
xlabel('Time (s)')
ylabel('Frequency')

for i = 1:length(I)
    hold on;
%   get the lag of index
    lagDiff = lag(I(i));
    timeDiff = lagDiff/Fs;
    plot(timeDiff*(-1), 1.0,'r*');
end
```

A.5 Normalised Cross-Correlation in 2d

```
% 3. Spatial Cross Correlation + Normalised Spatial Cross Correlation in 2d

% Images are just a matrix of grayscaled pixel values
% read the images.
image1 = fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment1','apple.png');
image2 = fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment1','small.jpg');
M1 = imread(image1); %search region
M2 = imread(image2); %template
M1 = rgb2gray(im2double(M1));%greyscale
M2 = rgb2gray(im2double(M2));%greyscale

% show the images
subplot(2,1,1)
imshow(M2);
title('template image')
subplot(2,1,2)
imshow(M1);
title('search region')

img = M1;% M1 is the search region (whole image)
Sec = M2;% M2 is the template (section)

% Subtract the mean value so that there are roughly equal numbers of negative and positive values.
```

```
nimg = img-mean(mean(img));
nSec = Sec-mean(mean(Sec));


% The 2-D Correlation block computes the two-dimensional cross-correlation of two input matrices.
% Assume that matrix A has dimensions (Ma, Na) and matrix B has dimensions (Mb, Nb).
% When the block calculates the full output size, the equation for the two-dimensional discrete ...
    cross-correlation is
% C(i,j) = (Ma-1)  m=0 (Na-1)  n=0 A(m,n).conj(B(m+i,n+j)), where 0 i <Ma+Mb-1 and 0 j <Na+Nb-1.

[M,N] = size(nimg);
m = 1:M;
n = 1:N;

[P,Q] = size(nSec);
p = 1:P;
q = 1:Q;

% Make a zero matrix and put the search region in the center
Xt = zeros([M+2*P N+2*Q]);
Xt(m+P,n+Q) = nimg;

% initialize the cross correlation matrix
C = zeros([M+P N+Q]);

for k = 1:M+P+1
    for l = 1:N+Q+1
        Xtl = Xt(p+k-1,q+l-1); %(1:P)+k-1,(1:Q)+l-1 because k and l start from 1 (should be 0).
        C(k,l) = ...
            sum(sum(Xtl.*conj(nSec)))/sqrt(sum(dot(Xtl,Xtl))*sum(dot(nSec,nSec)));%normalisation
    end
end

% trim NaN surrounding the matrix
C = C(2:(M+P),2:(N+Q));

C_test = normxcorr2(M2, M1);
figure
shading interp
surf(C);
title('normalised cross-correlation')
colorbar
```

A.6 Finding the Rocket Man

```
% 4. Where s Wally
tic
% read the images, make them grayscaled and double byte
image1 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment1','wallypuzzle_rocket_man.png');
image2 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment1','wallypuzzle_png.png');

M1 = imread(image1); %template
M2 = imread(image2); %search region
M1 = rgb2gray(im2double(M1));
M2 = rgb2gray(im2double(M2));
%imshow(M2);

% Subtract the mean value so that there are roughly equal numbers of negative and positive values.
nimg = M2-mean(mean(M2));
nSec = M1-mean(mean(M1));

[M,N] = size(nimg);
m = 1:M;
n = 1:N;

[P,Q] = size(nSec);
p = 1:P;
q = 1:Q;

% Make a zero matrix and put the search region in the center
Xt = zeros([M+2*P N+2*Q]);
```

```
Xt(m+P,n+Q) = nimg;

% initialize the cross correlation matrix
C = zeros([M+P N+Q]);

for k = 1:M+P+1
    for l = 1:N+Q+1
        Xtl = Xt(p+k-1,q+l-1); %(1:P)+k-1,(1:Q)+l-1 because k and l start from 1 (should be 0).
        C(k,l) = ...
            sum(sum(Xtl.*conj(nSec)))/sqrt(sum(dot(Xtl,Xtl))*sum(dot(nSec,nSec)));%normalisation
    end
end

% trim zeros around the matrix
C = C(2:(M+P),2:(N+Q));

% The maximum of the cross-correlation corresponds to the estimated location of the lower-right ...
    corner of the section.
% Use ind2sub to convert the one-dimensional location of the maximum to two-dimensional ...
    coordinates.
[ssr,snd] = max(C(:));
[ij,ji] = ind2sub(size(C),snd);

figure
plot(C(:))
title('Cross-Correlation')
xlabel('index')
ylabel('correlation value')
hold on
plot(snd,ssr,'or')
hold off
text(snd*1.05,ssr,'Maximum')
% draw the original search region image
figure
imshow(M2);
x1=ij-size(M1,1)+1;
x2=ij;
y1=ji-size(M1,2)+1;
y2=ji;

% draw the rectangle on the image
hold on;
x = [x1, x2, x2, x1, x1];
y = [y1, y1, y2, y2, y1];
plot(y, x, 'r');
plot(y1+(y2-y1)/2,x1+(x2-x1)/2, 'r*');%red star
hold off

toc
```

A.7 Dot Detection with Gausssian Template

```
% 1. Dot Detection Algorithm

% Read the image
image1 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','cal_image_left_2000.tiff');

M2 = rgb2gray(imread(image1));
% Gaussian template
M1 = [0,5,11,5,0
5,53,117,53,5
11,117,255,117,11
5,53,117,53,5
0,5,11,5,0];
surf(M1)
colorbar

% normalised cross correlation
C = normxcorr2(M1, M2);

% draw the original search region image
figure
```

```
title('Finding and plotting on the image')
% imagesc(M2)
% axis image off
% colormap gray
xlabel('x [mm]')
ylabel('y [mm]')
axis equal
axis([0 , 2500 , 0 , 1600]);
for i = 1:17*21
    [~,snd] = max(C(:));% get index of the high point in the vector
    [ij,ji] = ind2sub(size(C),snd);% convert the index to coordinates
    % get the x and y coordinates of the template zone
    x1=ij-size(M1,1)+1;
    x2=ij;
    y1=ji-size(M1,2)+1;
    y2=ji;
    hold on
    x = [x1, x2, x2, x1, x1];
    y = [y1, y1, y2, y2, y1];
    plot(y, x, 'r');
    C(ij-5:ij+5,ji-5:ji+5) = 0;% make the cross correlation zone of found dot to zero
end
```

A.8 Creating Calibration Model

```
% 2. Calibration Model
%% Preparation for real location matrices
% Create the X, Y, Z matrices for real locations
X = zeros(17,21,6);
Y = zeros(17,21,6);
Z = zeros(17,21,6);

% X and Y of the real location do not change
X_2000 = zeros(17,21);
Y_2000 = zeros(17,21);
for i = 1:17
    for j = 1:21
        X_2000(i,j) = -500 + (j-1)*50;
    end
end
for k = 1:6
    X(:,:,k) = X_2000;
end
for i = 1:17
    for j = 1:21
        Y_2000(i,j) = 800 - (i-1)*50;
    end
end
for k = 1:6
    Y(:,:,k) = Y_2000;
end

% The Z coordinate of the real location is changing
for i = 1:17
    for j = 1:21
        Z(i,j,1) = 2000;
    end
end
for i = 1:17
    for j = 1:21
        Z(i,j,2) = 1980;
    end
end
for i = 1:17
    for j = 1:21
        Z(i,j,3) = 1960;
    end
end
for i = 1:17
    for j = 1:21
        Z(i,j,4) = 1940;
    end
end
```

```
for i = 1:17
    for j = 1:21
        Z(i,j,5) = 1920;
    end
end
for i = 1:17
    for j = 1:21
        Z(i,j,6) = 1900;
    end
end

%% 1st pair of images
% Read the images for Z=2000mm, make them grayscaled and double byte
image1 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','cal_image_left_2000.tiff');
image2 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','cal_image_right_2000.tiff');
[i_left_2000,j_left_2000, i_right_2000, j_right_2000] = myfn(image1, image2);


%% 2nd pair of images
% Read the images for Z=1980mm, make them grayscaled and double byte
image1 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','cal_image_left_1980.tiff');
image2 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','cal_image_right_1980.tiff');
[i_left_1980,j_left_1980, i_right_1980, j_right_1980] = myfn(image1, image2);

%% 3rd pair of images
% Read the images for Z=1960mm, make them grayscaled and double byte
image1 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','cal_image_left_1960.tiff');
image2 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','cal_image_right_1960.tiff');
[i_left_1960,j_left_1960, i_right_1960, j_right_1960] = myfn(image1, image2);

%% 4th pair of images
% Read the images for Z=1940mm, make them grayscaled and double byte
image1 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','cal_image_left_1940.tiff');
image2 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','cal_image_right_1940.tiff');
[i_left_1940,j_left_1940, i_right_1940, j_right_1940] = myfn(image1, image2);

%% 5th pair of images
% Read the images for Z=1920mm, make them grayscaled and double byte
image1 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','cal_image_left_1920.tiff');
image2 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','cal_image_right_1920.tiff');
[i_left_1920,j_left_1920, i_right_1920, j_right_1920] = myfn(image1, image2);

%% 6th pair of images
% Read the images for Z=1900mm, make them grayscaled and double byte
image1 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','cal_image_left_1900.tiff');
image2 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','cal_image_right_1900.tiff');
[i_left_1900,j_left_1900, i_right_1900, j_right_1900] = myfn(image1, image2);

%% Ployfit the calibration model
% prepare for i and j matrices for all pairs of images
i_left = zeros(17,21,6);
j_left = zeros(17,21,6);
i_right = zeros(17,21,6);
j_right = zeros(17,21,6);
% put matrices of different pairs in
i_left(:,:,1) = i_left_2000;
i_left(:,:,2) = i_left_1980;
i_left(:,:,3) = i_left_1960;
i_left(:,:,4) = i_left_1940;
i_left(:,:,5) = i_left_1920;
i_left(:,:,6) = i_left_1900;

j_left(:,:,1) = j_left_2000;
j_left(:,:,2) = j_left_1980;
j_left(:,:,3) = j_left_1960;
j_left(:,:,4) = j_left_1940;
```

```
j_left(:,:,5) = j_left_1920;
j_left(:,:,6) = j_left_1900;

i_right(:,:,1) = i_right_2000;
i_right(:,:,2) = i_right_1980;
i_right(:,:,3) = i_right_1960;
i_right(:,:,4) = i_right_1940;
i_right(:,:,5) = i_right_1920;
i_right(:,:,6) = i_right_1900;

j_right(:,:,1) = j_right_2000;
j_right(:,:,2) = j_right_1980;
j_right(:,:,3) = j_right_1960;
j_right(:,:,4) = j_right_1940;
j_right(:,:,5) = j_right_1920;
j_right(:,:,6) = j_right_1900;

il = i_left(:);
jl = j_left(:);
ir = i_right(:);
jr = j_right(:);

% polyfit for X
x_input = [il , jl , ir , jr] ;
depvar_x = X(:);
x_fit = polyfitn(x_input, depvar_x, 3);
sx = polyn2sym(x_fit);

% polyfit for Y
y_input = [il , jl , ir , jr] ;
depvar_y = Y(:);
y_fit = polyfitn(y_input, depvar_y, 3);
sy = polyn2sym(y_fit);

% polyfit for Z
z_input = [il , jl , ir , jr] ;
depvar_z = Z(:);
z_fit = polyfitn(z_input, depvar_z, 3);
sz = polyn2sym(z_fit);

%% test for the pair of Z=2000mm images
%Get the calibration model
X = matlabFunction(sx);
Y = matlabFunction(sy);
Z = matlabFunction(sz);

Xreal_model = X(i_left_2000,j_left_2000,i_right_2000,j_right_2000);
Yreal_model = Y(i_left_2000,j_left_2000,i_right_2000,j_right_2000);
Zreal_model = Z(i_left_2000,j_left_2000,i_right_2000,j_right_2000);

%% function
function [i_left, j_left, i_right, j_right] = myfn(image1, image2)
i_left = zeros(1,17*21);
j_left = zeros(1,17*21);
i_right = zeros(1,17*21);
j_right = zeros(1,17*21);
% For left image
M2 = rgb2gray(imread(image1));
% Gaussian template
M1 = [0,5,11,5,0
5,53,117,53,5
11,117,255,117,11
5,53,117,53,5
0,5,11,5,0];

C = normxcorr2(M1, M2);
C(:,size(C,2)-4:size(C,2))=0;%remove edge effect
%------------------------------------------------------------------------
for i = 1:17*21
    [~,snd] = max(C(:));% get index of the high point in the vector
    [ij,ji] = ind2sub(size(C),snd);% convert the index to coordinates
    % get the x and y coordinates of the template zone
    x1=ij-size(M1,1)+1;
    x2=ij;
    y1=ji-size(M1,2)+1;
    y2=ji;
    j_left(i) = x1+(x2-x1)/2;% put the x coordinate into j_left.
    i_left(i) = y1+(y2-y1)/2;% put the y coordinate into i_left
```

```
    C(ij-5:ij+5,ji-5:ji+5) = 0;% make the cross correlation zone of found dot to zero
end
% sort and get the right order of coordinates
% 1-8 columns are in descending order
j_left = sort(j_left,'descend');
j_left = reshape(j_left,[21,17]);
j_left = transpose(j_left);
% 10-17 columns are in ascending order
j_left(10:17,:) = sort(j_left(10:17,:),2);
i_left = sort(i_left);
i_left = reshape(i_left,[17,21]);

%=========================================================================
% For right image
M2 = rgb2gray(imread(image2));
% Gaussian template
M1 = [0,5,11,5,0
5,53,117,53,5
11,117,255,117,11
5,53,117,53,5
0,5,11,5,0];

C = normxcorr2(M1, M2);
C(:,size(C,2)-4:size(C,2))=0;%remove edge effect
%-------------------------------------------------------------------------
for i = 1:17*21
    [~,snd] = max(C(:));% get index of the high point in the vector
    [ij,ji] = ind2sub(size(C),snd);% convert the index to coordinates
    % get the x and y coordinates of the template zone
    x1=ij-size(M1,1)+1;
    x2=ij;
    y1=ji-size(M1,2)+1;
    y2=ji;
    j_right(i) = x1+(x2-x1)/2;% put the x coordinate into j_left.
    i_right(i) = y1+(y2-y1)/2;% put the y coordinate into i_left
    C(ij-5:ij+5,ji-5:ji+5) = 0;% make the cross correlation zone of found dot to zero
end
% sort and get the right order of coordinates
% 10-17 columns are in descending order
j_right = sort(j_right,'descend');
j_right = reshape(j_right,[21,17]);
j_right = transpose(j_right);
% 1-8 columns are in ascending order
j_right(1:8,:) = sort(j_right(1:8,:),2);
i_right = sort(i_right);
i_right = reshape(i_right,[17,21]);
end
```

A.9 Test Scan on Dot Calibrated Images

```
% 5. Test Scan on Computer Generated Calibrated Images
% Should first run the part 2 for building the calibration model

% Read in a pair of images
image1 = fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','test_left_1.tiff');
image2 = fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','test_right_1.tiff');
%=========================================================================
% For left image
M2 = rgb2gray(imread(image1));
% Gaussian template
M1 = [0,5,11,5,0
5,53,117,53,5
11,117,255,117,11
5,53,117,53,5
0,5,11,5,0];

C = normxcorr2(M1, M2);

i_left_test = zeros(1,9*13);
j_left_test = zeros(1,9*13);
for i = 1:9*13
    [~,snd] = max(C(:));% get index of the high point in the vector
    [ij,ji] = ind2sub(size(C),snd);% convert the index to coordinates
```

```
    % get the x and y coordinates of the template zone
    x1=ij-size(M1,1)+1;
    x2=ij;
    y1=ji-size(M1,2)+1;
    y2=ji;
    j_left_test(i) = x1+(x2-x1)/2;% put the x coordinate into i_left.
    i_left_test(i) = y1+(y2-y1)/2;% put the y coordinate into j_left
    C(ij-5:ij+5,ji-5:ji+5) = 0;% make the cross correlation zone of found dot to zero
end
% sort and get the right order of coordinates
% 1-4 columns are in descending order
j_left_test = sort(j_left_test,'descend');
j_left_test = reshape(j_left_test,[13,9]);
j_left_test = transpose(j_left_test);
% 6-9 columns are in ascending order
j_left_test(6:9,:) = sort(j_left_test(6:9,:),2);
i_left_test = sort(i_left_test);
i_left_test = reshape(i_left_test,[9,13]);

%=========================================================================
% For right image
M2 = rgb2gray(imread(image2));
% Gaussian template
M1 = [0,5,11,5,0
5,53,117,53,5
11,117,255,117,11
5,53,117,53,5
0,5,11,5,0];

C = normxcorr2(M1, M2);

i_right_test = zeros(1,9*13);
j_right_test = zeros(1,9*13);
for i = 1:9*13
    [~,snd] = max(C(:));% get index of the high point in the vector
    [ij,ji] = ind2sub(size(C),snd);% convert the index to coordinates
    % get the x and y coordinates of the template zone
    x1=ij-size(M1,1)+1;
    x2=ij;
    y1=ji-size(M1,2)+1;
    y2=ji;
    j_right_test(i) = x1+(x2-x1)/2;% put the x coordinate into i_left.
    i_right_test(i) = y1+(y2-y1)/2;% put the y coordinate into j_left
    C(ij-5:ij+5,ji-5:ji+5) = 0;% make the cross correlation zone of found dot to zero
end
% sort and get the right order of coordinates
% 6-9 columns are in descending order
j_right_test = sort(j_right_test,'descend');
j_right_test = reshape(j_right_test,[13,9]);
j_right_test = transpose(j_right_test);
% 1-4 columns are in ascending order
j_right_test(1:4,:) = sort(j_right_test(1:4,:),2);
i_right_test = sort(i_right_test);
i_right_test = reshape(i_right_test,[9,13]);

%% Get the calibration model from part2_2
X = matlabFunction(sx);
Y = matlabFunction(sy);
Z = matlabFunction(sz);

Xreal_model = X(i_left_test,j_left_test,i_right_test,j_right_test);
Yreal_model = Y(i_left_test,j_left_test,i_right_test,j_right_test);
Zreal_model = Z(i_left_test,j_left_test,i_right_test,j_right_test);
%% Plot
dpx = i_left_test - i_right_test;
dpy = j_left_test - j_right_test;
figure
shading interp

subplot(2,1,1)
surf(dpx)
title('dpx')
xlabel('windows x')
ylabel('windows y')
zlabel('difference value')
colorbar

subplot(2,1,2)
```

```
surf(dpy)
title('dpy')
xlabel('windows x')
ylabel('windows y')
zlabel('difference value')
colorbar
%=========================================================================
figure;
plot3(Zreal_model,Yreal_model,Xreal_model,'oc')
hold on
shading interp
surf(reshape(Zreal_model,size(Zreal_model)),reshape(Yreal_model,size(Yreal_model)),reshape(Xreal_model,size(Xr
xlabel('z [mm]')
ylabel('y [mm]')
zlabel('x [mm]')
axis equal
z_min = 0;
z_max = 2000;
y_min = 0;
y_max = 800;
x_min = -500;
x_max = 500;
axis([z_min , z_max , y_min , y_max , x_min , x_max ]);
grid on
colorbar
```

A.10 Image Comparison

```
% 3. Image Comparison
% A big dx (big disparity between the found location in each camera) means the object is close ...
    to your cameras!
% A small dx means the object is further away.
% How far away? how close? That is what the calibration images are for.
%=========================================================================================================
tic
% Read in a pair of images
image1 = fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','left_portal.tiff');
image2 = fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','right_portal.tiff');
% test with the same image:
% image2 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','left_portal.tiff');


wsize = 64;% window size is 64*64
xgrid = 1+(wsize-1)/2;
ygrid = 1+(wsize-1)/2;

[dpx_all,dpy_all] = myfn(image1, image2, wsize, xgrid, ygrid);

figure
shading interp

subplot(2,1,1)
surf(dpx_all)
title('dpx')
xlabel('windows x')
ylabel('windows y')
zlabel('difference value')
colorbar

subplot(2,1,2)
surf(dpy_all)
xlabel('windows x')
ylabel('windows y')
zlabel('difference value')
colorbar
toc
%% function
function [dpx_all,dpy_all] = myfn(image1, image2, wsize, ~, ~)

M1_original = rgb2gray(imread(image1));
M2_original = rgb2gray(imread(image2));

```

```
% Get the number of windows (if the division cannot get integer)
tx = size(M1_original, 2);
ty = size(M1_original, 1);
x_windows = ceil(tx/wsize);
y_windows = ceil(ty/wsize);
% Divide the image into enough windows (3rd dimension is the number/index)
Template = zeros(wsize,wsize,x_windows*y_windows);
region = zeros(wsize*3,wsize*3,x_windows*y_windows);% 3x window size

% padding 0s to M1
M1 = zeros(y_windows*wsize, x_windows*wsize);
M1(1:ty,1:tx) = M1_original;
% padding 0s to M2
M2 = zeros(y_windows*wsize, x_windows*wsize);
M2(1:ty,1:tx) = M2_original;
% Break the left image into enough windows
for i = 1:y_windows
    for j = 1:x_windows
        Template(:,:,(i-1)*y_windows+j) = M1((i-1)*wsize+1:i*wsize,(j-1)*wsize+1:j*wsize);
    end
end

% generate a new search region with 0s surrounding the original region
M2_new = zeros(size(M2, 1)+wsize*2,size(M2, 2)+wsize*2);
M2_new(:,:) = 0;
M2_new(wsize+1:size(M2, 1)+wsize,wsize+1:size(M2, 2)+wsize) = M2;
% Break the corresponding windows in the right image
for i = 1:y_windows
    for j = 1:x_windows
        region(:,:,(i-1)*y_windows+j) = ...
            M2_new((i-1)*wsize+1:(i+3-1)*wsize,(j-1)*wsize+1:(j+3-1)*wsize);
    end
end

% 3 times means there is the same size at two sides, so the offset is 2 times of the window ...
    size from (0,0).
x_offset = wsize*2;
y_offset = wsize*2;
%=========================================================================
dpx_all = zeros(y_windows,x_windows);
dpy_all = zeros(y_windows,x_windows);
% Scan all the templates around the search region to find similar features using cross correlation
for i = 1:y_windows
    for j = 1:x_windows
        template = Template(:,:,(i-1)*y_windows+j);
        background = region(:,:,(i-1)*y_windows+j);

        % cross correlation
        C = normxcorr2(template, background);
        [~,snd] = max(C(:));% get index of the high point in the vector
        [ij,ji] = ind2sub(size(C),snd);% convert the index to coordinates
        dpx_all(i,j) = ji-x_offset;
        dpy_all(i,j) = ij-y_offset;
    end
end
end
```

A.11 Test Scan on General Calibrated Images

```
% 5. Test Scan on Computer Generated Calibrated Images
% Should first run the part 2 for building the calibration model
tic
% Read in a pair of images
image1 = fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','test_left_2.tiff');
image2 = fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','test_right_2.tiff');

wsize = 64;% window size is 64*64
xgrid = 1+(wsize-1)/2;% x pixel of the center point
ygrid = 1+(wsize-1)/2;% y pixel of the center point
region_time = 5;

[dpx_all,dpy_all,i_left_test,j_left_test,x_windows,y_windows] = myfn(image1, image2, wsize, ...
    xgrid, ygrid, 1, 5);
```

```
i_right_test = i_left_test + dpx_all;
j_right_test = j_left_test + dpy_all;

figure
shading interp

subplot(2,1,1)
surf(dpx_all)
title('dpx')
xlabel('windows x')
ylabel('windows y')
zlabel('difference value')
colorbar

subplot(2,1,2)
surf(dpy_all)
title('dpy')
xlabel('windows x')
ylabel('windows y')
zlabel('difference value')
colorbar
%=========================================================================
% % Remove spurious vectors larger than 50
% for i = 1:y_windows
%     for j = 1:x_windows
%         if(abs(dpx_all(i,j)) > 50)
%             dpx_all(i,j) = NaN;
%         end
%     end
% end
% for i = 1:y_windows
%     for j = 1:x_windows
%         if(abs(dpy_all(i,j)) > 50)
%             dpy_all(i,j) = NaN;
%         end
%     end
% end
%
% figure
% shading interp
%
% subplot(2,1,1)
% surf(dpx_all)
% title('dpx after removing spurious vectors')
% xlabel('windows x')
% ylabel('windows y')
% zlabel('difference value')
% colorbar
%
% subplot(2,1,2)
% surf(dpy_all)
% title('dpy after removing spurious vectors')
% xlabel('windows x')
% ylabel('windows y')
% zlabel('difference value')
% colorbar

%% Get real models using calibration model
X = matlabFunction(sx);
Y = matlabFunction(sy);
Z = matlabFunction(sz);

Xreal_model = X(i_left_test,j_left_test,i_right_test,j_right_test);
Yreal_model = Y(i_left_test,j_left_test,i_right_test,j_right_test);
Zreal_model = Z(i_left_test,j_left_test,i_right_test,j_right_test);
% Plot 3D construction
figure;
%plot3(Zreal_model,Yreal_model,Xreal_model,'oc')
hold on
surf(reshape(Zreal_model,size(Zreal_model)),reshape(Yreal_model,size(Yreal_model)),reshape(Xreal_model,size(Xr
xlabel('z [mm]')
ylabel('y [mm]')
zlabel('x [mm]')
axis equal
z_min = 0;
z_max = 2200;
y_min = -100;
y_max = 500;
```

```
x_min = -800;
x_max = 800;
axis([z_min , z_max , y_min , y_max , x_min , x_max ]);
grid on
colorbar

toc
%% function
% see xgrid is centre location of x1 and x2,
% xgrid = x1 + (x2-x1)/2
% xgrid = x1 + (wsize-1)/2
% so x1 = xgrid - (wsize-1)/2
function [dpx_all,dpy_all,i_left_test,j_left_test, x_windows, y_windows] = myfn(image1, image2, ...
    wsize, xgrid, ygrid, overlap, region_time)
%function [dpx_all,dpy_all] = myfn(image1, image2, wsize, ~, ~, overlap, region_time)

M1_original = rgb2gray(imread(image1));
M2_original = rgb2gray(imread(image2));

% Get the number of windows
tx = size(M1_original, 2);
ty = size(M1_original, 1);
x_windows = ceil(tx/wsize);
y_windows = ceil(ty/wsize);
% Divide the image into enough windows
Template = zeros(wsize,wsize,x_windows*(1/overlap) * y_windows*(1/overlap));% 1/overlap = 2
region = zeros(wsize*region_time,wsize*region_time,x_windows*(1/overlap) * ...
    y_windows*(1/overlap));% 5x window size

% padding enough 0s to M1, the last window will be shifted outside, should multiplied by 2 for ...
    convenience.
M1 = zeros(ty*2, tx*2);
M1(1:ty,1:tx) = M1_original;
% padding enough 0s to M2
M2 = zeros(ty*2, tx*2);
M2(1:ty,1:tx) = M2_original;
% Break the left image into enough windows
for i = 1:y_windows*(1/overlap)
    for j = 1:x_windows*(1/overlap)
        Template(:,:,(i-1)*y_windows*(1/overlap)+j) = ...
            M1((i-1)*(wsize*overlap)+1:wsize*(i*overlap-overlap+1),(j-1)*(wsize*overlap)+1:wsize*(j*overlap-ov
    end
end

% generate a new search region with 0s surrounding the original region
M2_new = zeros(size(M2, 1)+wsize*(region_time-1),size(M2, 2)+wsize*(region_time-1));
M2_new(:,:) = 0;
M2_new(wsize*(region_time-1)/2+1:size(M2, ...
    1)+wsize*(region_time-1)/2,wsize*(region_time-1)/2+1:size(M2, 2)+wsize*(region_time-1)/2) = M2;
% Break the corresponding windows in the right image
for i = 1:y_windows*(1/overlap)
    for j = 1:x_windows*(1/overlap)
        region(:,:,(i-1)*y_windows*(1/overlap)+j) = ...
            M2_new((i-1)*(wsize*overlap)+1:wsize*(i*overlap-overlap+region_time),(j-1)*(wsize*overlap)+1:wsize
    end
end

% The maximum of the cross-correlation corresponds to the
% estimated location of the lower-right corner of the template on background.
% like 5 times means there is the double size at two sides,
% so the offset is 3 times of the window size from (0,0).
x_offset = wsize*((region_time-1)/2+1);
y_offset = wsize*((region_time-1)/2+1);
%=========================================================================

%=========================================================================
dpx_all = zeros(y_windows*(1/overlap),x_windows*(1/overlap));
dpy_all = zeros(y_windows*(1/overlap),x_windows*(1/overlap));
i_left_test = zeros(y_windows,x_windows);
j_left_test = zeros(y_windows,x_windows);
% Scan all the templates around the search region to find similar features using cross correlation
for i = 1:y_windows*(1/overlap)
    for j = 1:x_windows*(1/overlap)
        template = Template(:,:,(i-1)*y_windows*(1/overlap)+j);
        background = region(:,:,(i-1)*y_windows*(1/overlap)+j);

        % cross correlation
        C = normxcorr2(template, background);
```

```
        [~,snd] = max(C(:));% get index of the high point in the vector
        [ij,ji] = ind2sub(size(C),snd);% convert the index to coordinates
        dpx_all(i,j) = ji-x_offset;
        dpy_all(i,j) = ij-y_offset;
        i_left_test(i,j) = j*xgrid;
        j_left_test(i,j) = i*ygrid;

    end
end
end
```

A.12 Image Comparison Optimisation

```
% 4. Cross Correlation Optimisation
% a) Window Overlap = 50%
% b) Search region = 5x Window Size, Square

%=========================================================================================================
% Read in a pair of images
image1 = fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','left_portal.tiff');
image2 = fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','right_portal.tiff');

wsize = 64;% window size is 64*64
% center pixel of the first window
xgrid = 1+(wsize-1)/2;
ygrid = 1+(wsize-1)/2;
% a) Window overlap = 50%, default is 1
overlap = 0.5;
% b) Search region = 5x Window Size, Square, default is 3
region_time = 5;

%=========================================================================
tic
[dpx,dpy] = myfn(image1, image2, wsize, xgrid, ygrid, overlap, 3);
figure
shading interp

subplot(2,1,1)
surf(dpx)
title('dpx when overlap is 50%')
xlabel('windows x')
ylabel('windows y')
zlabel('difference value')
colorbar

subplot(2,1,2)
surf(dpy)
title('dpy when overlap is 50%')
xlabel('windows x')
ylabel('windows y')
zlabel('difference value')
colorbar
toc
%=========================================================================
tic
[dpx,dpy] = myfn(image1, image2, wsize, xgrid, ygrid, 1, region_time);
figure
shading interp

subplot(2,1,1)
surf(dpx)
title('dpx when search region is 5 times bigger')
xlabel('windows x')
ylabel('windows y')
zlabel('difference value')
colorbar

subplot(2,1,2)
surf(dpy)
xlabel('windows x')
ylabel('windows y')
zlabel('difference value')
title('dpy when search region is 5 times bigger')
```

```
colorbar
toc

%=========================================================================
tic
% default function for first pass
[dpx_est,dpy_est] = myfn(image1, image2, wsize, xgrid, ygrid, 1, 3);
% second pass
[dpx,dpy] = myfn2(image1, image2, wsize, xgrid, ygrid, dpx_est, dpy_est);
figure
shading interp

subplot(2,1,1)
surf(dpx)
title('dpx when second pass')
xlabel('windows x')
ylabel('windows y')
zlabel('difference value')
colorbar

subplot(2,1,2)
surf(dpy)
title('dpy when second pass')
xlabel('windows x')
ylabel('windows y')
zlabel('difference value')
colorbar
toc
%% function
% see xgrid is centre location of x1 and x2,
% xgrid = x1 + (x2-x1)/2
% xgrid = x1 + (wsize-1)/2
% so x1 = xgrid - (wsize-1)/2
function [dpx_all,dpy_all] = myfn(image1, image2, wsize, ~, ~, overlap, region_time)

M1_original = rgb2gray(imread(image1));
M2_original = rgb2gray(imread(image2));

% Get the number of windows
tx = size(M1_original, 2);
ty = size(M1_original, 1);
x_windows = ceil(tx/wsize);
y_windows = ceil(ty/wsize);
% Divide the image into enough windows
Template = zeros(wsize,wsize,x_windows*(1/overlap) * y_windows*(1/overlap));% 1/overlap = 2
region = zeros(wsize*region_time,wsize*region_time,x_windows*(1/overlap) * ...
    y_windows*(1/overlap));% 5x window size

% Padding enough 0s to M1.
% The last window may be shifted outside, could multiply size by 2 for convenience.
M1 = zeros(ty*2, tx*2);
M1(1:ty,1:tx) = M1_original;
% Padding enough 0s to M2
M2 = zeros(ty*2, tx*2);
M2(1:ty,1:tx) = M2_original;

% Break the left image into enough windows
for i = 1:y_windows*(1/overlap)
    for j = 1:x_windows*(1/overlap)
        Template(:,:,(i-1)*y_windows*(1/overlap)+j) = ...
            M1((i-1)*(wsize*overlap)+1:wsize*(i*overlap-overlap+1),(j-1)*(wsize*overlap)+1:wsize*(j*overlap-ov
    end
end

% Generate a new search region with 0s surrounding the original region
M2_new = zeros(size(M2, 1)+wsize*(region_time-1),size(M2, 2)+wsize*(region_time-1));
M2_new(:,:) = 0;
M2_new(wsize*(region_time-1)/2+1:size(M2, ...
    1)+wsize*(region_time-1)/2,wsize*(region_time-1)/2+1:size(M2, 2)+wsize*(region_time-1)/2) = M2;

% Break the corresponding windows in the right image
for i = 1:y_windows*(1/overlap)
    for j = 1:x_windows*(1/overlap)
        region(:,:,(i-1)*y_windows*(1/overlap)+j) = ...
            M2_new((i-1)*(wsize*overlap)+1:wsize*(i*overlap-overlap+region_time),(j-1)*(wsize*overlap)+1:wsize
    end
end

```

```
% 5 times means there is the double size at two sides,
% so the offset is 3 times of the window size from (0,0).
x_offset = wsize*((region_time-1)/2+1);
y_offset = wsize*((region_time-1)/2+1);
%=========================================================================

%=========================================================================
dpx_all = zeros(y_windows*(1/overlap),x_windows*(1/overlap));
dpy_all = zeros(y_windows*(1/overlap),x_windows*(1/overlap));
% Scan all the templates around the search region to find similar features using cross correlation
for i = 1:y_windows*(1/overlap)
    for j = 1:x_windows*(1/overlap)
        template = Template(:,:,(i-1)*y_windows*(1/overlap)+j);
        background = region(:,:,(i-1)*y_windows*(1/overlap)+j);

        % cross correlation
        C = normxcorr2(template, background);
        [~,snd] = max(C(:));% get index of the high point in the vector
        [ij,ji] = ind2sub(size(C),snd);% convert the index to coordinates
        dpx_all(i,j) = ji-x_offset;
        dpy_all(i,j) = ij-y_offset;

    end
end
end

%% function 2 - 2nd pass
function [dpx_all,dpy_all] = myfn2(image1, image2, wsize, ~, ~, dpx_est, dpy_est)
M1_original = rgb2gray(imread(image1));
M2_original = rgb2gray(imread(image2));

% Get the number of windows
tx = size(M1_original, 2);
ty = size(M1_original, 1);
x_windows = ceil(tx/wsize);
y_windows = ceil(ty/wsize);
% Divide the image into enough windows
Template = zeros(wsize,wsize,x_windows*y_windows);
region = zeros(wsize,wsize,x_windows*y_windows);% 1x window size

% padding 0s to M1
M1 = zeros(y_windows*wsize, x_windows*wsize);
M1(1:ty,1:tx) = M1_original;
% padding 0s to M2
M2 = zeros(y_windows*wsize, x_windows*wsize);
M2(1:ty,1:tx) = M2_original;
% Break the left image into enough windows
for i = 1:y_windows
    for j = 1:x_windows
        Template(:,:,(i-1)*y_windows+j) = M1((i-1)*wsize+1:i*wsize,(j-1)*wsize+1:j*wsize);
    end
end

% generate a new search region with enough 0s surrounding the original region
M2_new = zeros(size(M2, 1)+wsize*4,size(M2, 2)+wsize*4);
M2_new(:,:) = 0;
M2_new(wsize*2+1:size(M2, 1)+wsize*2,wsize*2+1:size(M2, 2)+wsize*2) = M2;
% Break the corresponding windows in the right image
for i = 1:y_windows
    for j = 1:x_windows
        region(:,:,(i-1)*y_windows+j) = ...
            M2_new((i-1)*wsize+wsize*2+1+dpy_est(i,j):i*wsize+wsize*2+dpy_est(i,j),(j-1)*wsize+wsize*2+1+dpx_e
    end
end

%=========================================================================
dpx_all = zeros(y_windows*2,x_windows*2);
dpy_all = zeros(y_windows*2,x_windows*2);
% Scan all the templates around the search region to find similar features using cross correlation
% 2nd pass is 32*32 (more detailed)
for i = 1:y_windows
    for j = 1:x_windows
        template = Template(:,:,(i-1)*y_windows+j);
        background = region(:,:,(i-1)*y_windows+j);

        % dpx/dpy is the relative coordinate substract the offset of
        % the part, and then plus the estimated dpx/dpy.

```

```
212         % 1.cross correlation for top-left part
213         C = normxcorr2(template(1:wsize/2,1:wsize/2), background);
214         [~,snd] = max(C(:));% get index of the high point in the vector
215         [ij,ji] = ind2sub(size(C),snd);% convert the index to coordinates
216 
217         dpx_all((i-1)*2+1,(j-1)*2+1) = ji-wsize/2+dpx_est(i,j);
218         dpy_all((i-1)*2+1,(j-1)*2+1) = ij-wsize/2+dpy_est(i,j);
219 
220         % 2.cross correlation for top-right part
221         C = normxcorr2(template(1:wsize/2,1+wsize/2:wsize), background);
222         [~,snd] = max(C(:));% get index of the high point in the vector
223         [ij,ji] = ind2sub(size(C),snd);% convert the index to coordinates
224         dpx_all((i-1)*2+1,j*2) = ji-wsize+dpx_est(i,j);
225         dpy_all((i-1)*2+1,j*2) = ij-wsize/2+dpy_est(i,j);
226 
227         % 3.cross correlation for bot-left part
228         C = normxcorr2(template(wsize/2+1:wsize,1:wsize/2), background);
229         [~,snd] = max(C(:));% get index of the high point in the vector
230         [ij,ji] = ind2sub(size(C),snd);% convert the index to coordinates
231         dpx_all(i*2,(j-1)*2+1) = ji-wsize/2+dpx_est(i,j);
232         dpy_all(i*2,(j-1)*2+1) = ij-wsize+dpy_est(i,j);
233 
234         % 4.cross correlation for bot-right part
235         C = normxcorr2(template(wsize/2+1:wsize,1+wsize/2:wsize), background);
236         [~,snd] = max(C(:));% get index of the high point in the vector
237         [ij,ji] = ind2sub(size(C),snd);% convert the index to coordinates
238         dpx_all(i*2,j*2) = ji-wsize+dpx_est(i,j);
239         dpy_all(i*2,j*2) = ij-wsize+dpy_est(i,j);
240 
241     end
242 end
243 end
244 %% Self-implemented normalised cross correlation function
245 function C = selfnormxcorr2(M2, M1)
246 img = M1;% M1 is the search region (whole image)
247 Sec = M2;% M2 is the template (section)
248 
249 % Subtract the mean value so that there are roughly equal numbers of negative and positive values.
250 nimg = img-mean(mean(img));
251 nSec = Sec-mean(mean(Sec));
252 
253 [M,N] = size(nimg);
254 m = 1:M;
255 n = 1:N;
256 
257 [P,Q] = size(nSec);
258 p = 1:P;
259 q = 1:Q;
260 
261 % Make a zero matrix and put the search region in the center
262 Xt = zeros([M+2*P N+2*Q]);
263 Xt(m+P,n+Q) = nimg;
264 
265 % initialize the cross correlation matrix
266 C = zeros([M+P N+Q]);
267 
268 for k = 1:M+P+1
269     for l = 1:N+Q+1
270         Xtl = Xt(p+k-1,q+l-1); %(1:P)+k-1,(1:Q)+l-1 because k and l start from 1 (should be 0).
271         C(k,l) = ...
                sum(sum(Xtl.*conj(nSec)))/sqrt(sum(dot(Xtl,Xtl))*sum(dot(nSec,nSec)));%normalisation
272     end
273 end
274 
275 % trim NaN surrounding the matrix
276 C = C(2:(M+P),2:(N+Q));
277 end
```

A.13 Sub-pixel Implementation

```
1 % 3. Image Comparison
2 % A big dx (big disparity between the found location in each camera) means the object is close ...
      to your cameras!
```

```
% A small dx means the object is further away.
% How far away? how close? That is what the calibration images are for.
%=========================================================================================================
tic
% Read in a pair of images
image1 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','small_left_portal.png');
image2 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','small_right_portal.png');
% test with the same image:
% image2 = ...
    fullfile('C:\','Users','frankie','Desktop','COMP90072','assignment2','left_portal.tiff');


wsize = 6.4*10;% window size is 6.4*6.4

[dpx_all,dpy_all] = myfn(image1, image2, wsize);

figure
shading interp

subplot(2,1,1)
surf(dpx_all)
title('dpx')
xlabel('windows x')
ylabel('windows y')
zlabel('difference value')
colorbar

subplot(2,1,2)
surf(dpy_all)
xlabel('windows x')
ylabel('windows y')
zlabel('difference value')
colorbar
toc
%% function
function [dpx_all,dpy_all] = myfn(image1, image2, wsize)

M1_original = im2double(rgb2gray(imread(image1)));
M2_original = im2double(rgb2gray(imread(image2)));

% sub-pixel
% Converting from images to meshgrid
% Create the query grid with spacing of 0.1.
% Interpolate at the query points.
[X1,Y1] = meshgrid(1:size(M1_original,2), 1:size(M1_original,1));
V1 = M1_original;
[Xq1,Yq1] = meshgrid(1:0.1:size(M1_original,2),1:0.1:size(M1_original,1));
Vq1 = interp2(X1,Y1,V1,Xq1,Yq1);

[X2,Y2] = meshgrid(1:size(M2_original,2), 1:size(M2_original,1));
V2 = M2_original;
[Xq2,Yq2] = meshgrid(1:0.1:size(M2_original,2),1:0.1:size(M2_original,1));
Vq2 = interp2(X2,Y2,V2,Xq2,Yq2);

% Get the number of windows (if the division cannot get integer)
tx = size(Vq1, 2);
ty = size(Vq1, 1);
x_windows = ceil(tx/wsize);
y_windows = ceil(ty/wsize);
% Divide the image into enough windows (3rd dimension is the number/index)
Template = zeros(wsize,wsize,x_windows*y_windows);
region = zeros(wsize*3,wsize*3,x_windows*y_windows);% 3x window size

% padding 0s to M1
M1 = zeros(y_windows*wsize, x_windows*wsize);
M1(1:ty,1:tx) = Vq1;
% padding 0s to M2
M2 = zeros(y_windows*wsize, x_windows*wsize);
M2(1:ty,1:tx) = Vq2;
% Break the left image into enough windows
for i = 1:y_windows
    for j = 1:x_windows
        Template(:,:,(i-1)*y_windows+j) = M1((i-1)*wsize+1:i*wsize,(j-1)*wsize+1:j*wsize);
    end
end

```

```
% generate a new search region with 0s surrounding the original region
M2_new = zeros(size(M2, 1)+wsize*2,size(M2, 2)+wsize*2);
M2_new(:,:) = 0;
M2_new(wsize+1:size(M2, 1)+wsize,wsize+1:size(M2, 2)+wsize) = M2;
% Break the corresponding windows in the right image
for i = 1:y_windows
    for j = 1:x_windows
        region(:,:,(i-1)*y_windows+j) = ...
            M2_new((i-1)*wsize+1:(i+3-1)*wsize,(j-1)*wsize+1:(j+3-1)*wsize);
    end
end

% 3 times means there is the same size at two sides, so the offset is 2 times of the window ...
    size from (0,0).
x_offset = wsize*2;
y_offset = wsize*2;
%=========================================================================
dpx_all = zeros(y_windows,x_windows);
dpy_all = zeros(y_windows,x_windows);
% Scan all the templates around the search region to find similar features using cross correlation
for i = 1:y_windows
    for j = 1:x_windows
        template = Template(:,:,(i-1)*y_windows+j);
        background = region(:,:,(i-1)*y_windows+j);

        % cross correlation
        C = normxcorr2(template, background);
        [~,snd] = max(C(:));% get index of the high point in the vector
        [ij,ji] = ind2sub(size(C),snd);% convert the index to coordinates
        dpx_all(i,j) = ji-x_offset;
        dpy_all(i,j) = ij-y_offset;
    end
end
end
```

Bibliography

[1] T. S. Archive. *Sounds from the Star Wars Movies*. http://www.thesoundarchive.com/star-wars.asp.

[2] N. Audio. *"How an FFT works"*. www.nti-audio.com.

[3] L. Newson USA. *Brass Calibration Plates*. http://www.newsonusallc.com/services/calibration-media/brass-calibration-plates/.

[4] U. of Melbourne. *Stereo Vision abstract*. 2018.