



THE UNIVERSITY OF
MELBOURNE

FINAL REPORT

The Art of Scientific Computing: Stereo Vision

Author:
Haonan Li

Subject Handbook code:
COMP-90072

The University of Melbourne

10 April 2018

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

Contents

1	Introduction	2
2	Cross Correlations	3
2.1	Normalized Spatial Cross Correlation in 1d	3
2.2	Signal Offset	4
2.3	Normalised Spatial Cross Correlation in 2d	4
2.4	Image Alignment	5
2.5	Spectral Cross Correlation	5
2.6	Pattern Finder	7
2.7	Cross correlation in three dimensions	7
3	Calibration Model	8
3.1	Dot Detection Algorithm	8
3.2	Create Calibration Model	9
3.3	Image Comparision	10
3.4	Cross Correlation Optimization	12
3.5	Test Scan on Computer Generated Calibrated Images	14
3.6	Optimized Test Scan	14
3.7	Better Calibration model	14
4	Depth Map Prediction using Machine Learning	16
4.1	Convolutional Neural Network	16
4.2	Stereo ConvNet	18
Appendix A	Matlab code	19
A.1	Part 1	19
A.2	Part 2	23
Bibliography		32

Chapter 1

Introduction

As we all know, organisms today are products of natural selection, which eliminates inferior species gradually over time. In the mean time, It is clearly obvious that most animals have two eyes, referred to as Binocular vision. What advantages of this biological structure? One good reason is that having two eyes provides a much wider field of view. However, research shows that human humans have a maximum horizontal field of view of 188 degrees with two eyes but 124 degrees of this is the binocular field of view (seen by both eyes). Why so much overlap of each field of view rather than less overlap and wider field of uni-ocular fields (seen by only one eye)?

The answer is having two eyes gives us perception of depth and 3-dimensional structure obtained on the basis of visual information deriving from them, which is referred to as stereopsis. Because the eyes of humans and many animals are located at different lateral positions on the head, binocular vision results in two slightly different images projected to the retinas of the eyes. The differences are mainly in horizontal position in two images, refer to binocular disparities. Our brain process these informations and gives precise depth perception.

In this project. We investigate how Binocular works by implementing a stereo vision model. In stereo vision, two cameras instead of eyes are displaced horizontally from one another, and used to obtain two differing views on a scene, in a manner similar to human binocular vision. By comparing these two images, we can extract relative depth information in the form of a disparity map, which encodes the difference in horizontal coordinates of corresponding image points. The values in this disparity map are inversely proportional to the scene depth at the corresponding pixel location.

In Chapter 2, we implement spatial cross correlation computation model. In the first two sections, we build a one dimension cross correlation model, which is very useful in signal processing and we apply it to a signal offset compute. In the next two sections, we extend our model to two dimensions, makes it work in image comparison and apply it to a particular pattern search task. Cause spatial cross correlation is time consuming, we want to find a faster method to achieve same computational ability of cross correlation, we explore and implement spectral cross correlation in section 5, and compare the run time of two methods. The next section is a application of 1d cross correlation, we apply this model to pattern detection. And the last section is our thinking and future work about cross correlation.

In Chapter 3, the image comparison model is build and we illustrate the depth perception use the model. We first create a calibration model use calibration plant in first two sections. We then explore how to correct images for parallax and other distortions, cross calibrate images and use this information to create a mapping algorithm to extract depth data from two planar images in section 3,4,5, meanwhile, we develop some optimization strategies to the model, some of them are design for speed and some for accuracy improvement.

Chapter 2

Cross Correlations

Cross correlation is powerful (and very simple) statistical tool for computing the degree to which two signals are correlated (or similar), and also for computing lags. In this part, we first implement the cross correlation in 1 dimension and apply it to a signal processing task. Then we extend it to 2 dimensions and do some image processing with it.

After implementing these basic models, we focus on the efficiency of compute cross correlation and find it is time consumed as the signals and images become larger. We then find spectral cross correlation with a Fourier transform and inverse Fourier transform could be an alternative way. The result shows this new method achieve better performance in scale data.

2.1 Normalized Spatial Cross Correlation in 1d

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. In this project, we only consider discrete signals rather than continue signals.

For any one dimension signal, we can represent it by a function. For discrete functions f and g , the cross-correlation is defined as:

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

Which computes the correlation between two signals. However, the value of r is somewhat arbitrary because of the variant amplitude of f and g , which makes the position of maximum value of r does not represents the offset that makes two signals get maximum similarity. One way around this is to normalize signal with the root-mean-square. Normalized cross correlation is typically done by subtracting the mean and dividing by the standard deviation. As:

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

where

$$\sigma_f = \sqrt{\frac{1}{N} \sum_{i=1}^{i=N} (f(i) - \bar{f})^2} \quad \sigma_g = \sqrt{\frac{1}{N} \sum_{i=1}^{i=N} (g(i) - \bar{g})^2}$$

Source code of one dimensional cross correlation and normalized one dimensional cross correlation are in Appendix A.1.1 and A.1.2 separately.

2.2 Signal Offset

Consider two signals (denote by vector) come from the same source and they are just offset by some time. We can use cross correlation find the offset time and distance between two sensors easily. First find the max value's position of the cross correlation vector computed from two signals. Then find corresponding positions of the two signals and compute the offset of the signal vector. If we know the sample rate and propagation speed, we can compute offset time and distance between the two sensors as follow:

$$\text{offset time} = \frac{\text{offset}}{\text{sample rate}}$$

$$\text{distance} = (\text{offset time}) * (\text{propagation speed})$$

In this task, we take two signal files and compute the cross correlation between them. Figure 2.1 shows the result of the cross correlation.

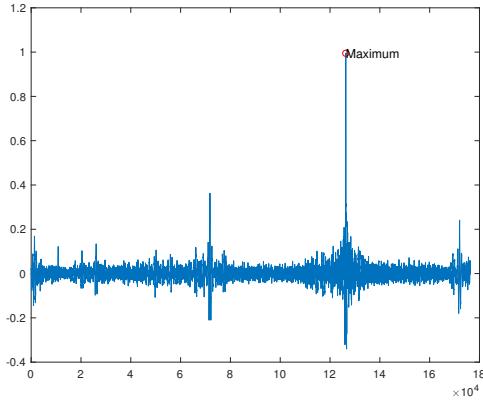


Figure 2.1: Cross correlation of two signals.

The x coordinate of maximum correlation is 50081, the length of signal is 176401 and sample rate is 44100. Apply the function above and get the result of offset time equals 50081 and sensor distance equals 378.17 meters.

Source code of signal offset computing is in Appendix A.1.3.

2.3 Normalised Spatial Cross Correlation in 2d

Till now, the cross correlation we used can only computer one dimensional signal. But in later work, we want to compare images and get the correlation of them, so we must extend the cross correlation to two dimensions. Similar with signals can be represent by vector, images can represented by matrix. Consider two matrices, t (template) and A (search region), The matrix A will always larger than the matrix t . We can use two nested for-loops to “leg” t over A , compute for each “lag” the cross-correlation. Normalized cross correlation of two matrices defines as:

$$R(lag_x, lag_y) = \frac{\sum_{x,y} [A(x, y) - \bar{A}_{lag_x, lag_y}] [t(x - lag_x, y - lag_y) - \bar{t}]}{\{\sum_{x,y} [A(x, y) - \bar{A}_{lag_x, lag_y}]^2\}^{0.5} \sum_{x,y} [t(x - lag_x, y - lag_y) - \bar{t}]^2}$$

Where \bar{t} is the mean of t , \bar{A}_{lag_x, lag_y} is the mean of A in the region under t .

The source code of implementation of normalized 2d cross correlation is in Appendix A.1.4.

2.4 Image Alignment

Images are just a matrix of pixel values in most image processing. It is naturally think of computing cross correlation between two images to get more relation informations of them. In this task, we get two images, one of them is a section of the other. We can easily find where the section of the image fits in the whole through cross correlation.

Here, we have a section (rocket man) show in Figure 2.2a and a search region (maze) as Figure 2.2b.

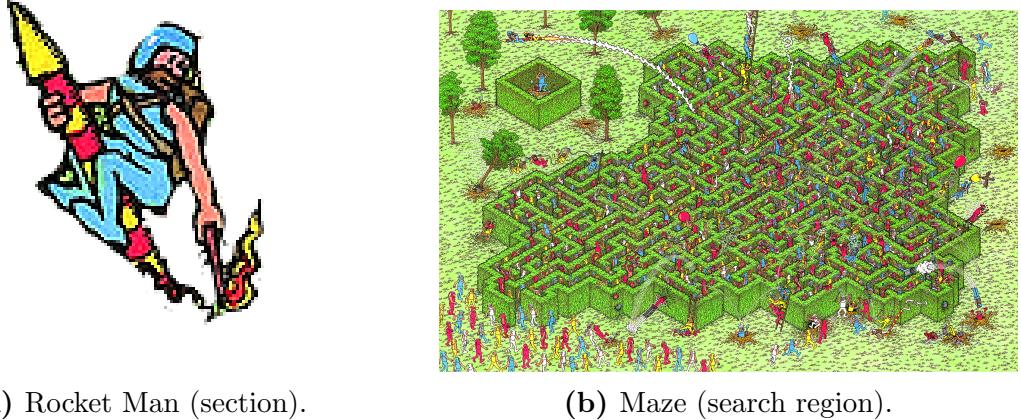


Figure 2.2: Source images for image alignment.

The visualization of cross correlation is shown in Figure 2.3a and Figure 2.3b . The maximum of the cross correlation corresponds to the estimated location of the section. Which means the most similar position of the section and search region. We mark it with a red star in the whole image. The result shows in Figure 2.4.

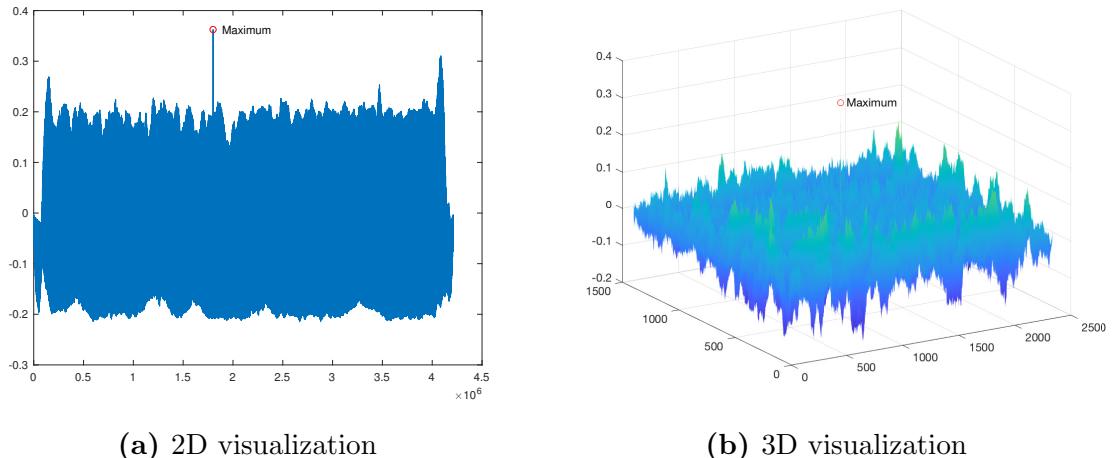


Figure 2.3: Cross correlation result matrix visualization

Source code for image alignment is in Appendix A.1.5.

2.5 Spectral Cross Correlation

Cross correlation can also be done in the spectral domain by completing a Fourier transform, multiplying signals, and doing an inverse Fourier transform.



Figure 2.4: Image Alignment result.

In this task, we use a fast Fourier transform (FFT) algorithm to convert a signal from its original domain to a representation in the frequency domain and inverse fast Fourier transform (IFFT) vice versa. It is an advanced algorithm of the discrete Fourier transform (DFT), And it manages to reduce the complexity of computing the DFT from $O(n^2)$, which arises if one simply applies the definition of DFT, to $O(n \log n)$.

Fast Fourier transform is a very useful and powerful algorithm in computing convolution and cross correlation. It can be shown that the discrete convolution of signal u and v as defined by,

$$(u * v)(\tau) = \sum_{m=1}^N u(m)v(\tau - m)$$

can also be expressed in terms of Fourier transform

$$(u * v)(\tau) = \mathcal{F}^{-1}\{\mathcal{F}(u) \cdot \mathcal{F}(v)\}$$

where $\mathcal{F}(u)$ is the Fourier transform of u , $\mathcal{F}(v)$ is the Fourier transform of v , and \mathcal{F}^{-1} is the inverse Fourier transform.

As for cross correlation, it can be calculated through summation of a product.

$$(u * v)(\tau) = \sum_{m=1}^N u^*(m)v(\tau + m)$$

or using FFTs.

$$(u * v)(\tau) = \mathcal{F}^{-1}\{(\mathcal{F}(u))^* \cdot \mathcal{F}(v)\}$$

Where the $*$ refers to the complex conjugate.

Table 2.1 compare the run time of two different method of task in section 2. From which we can find that spectral method using Fourier transform achieves better performance.

Table 2.1: Run time compare of spatial and spectral methods.

method	Spatial	Spectral
time(s)	4.6741	0.1103

Source code of spectral cross correlation is in Appendix A.1.6.

2.6 Pattern Finder

In this task, We pick a piece of music and cut a small piece of it (a drum) through listening. Figure 2.5 shows two pieces of waveforms separately. And 2.5c shows all occurrence of the element, because it is not a normalized correlation picture, all positions with the y-coordinates larger than 1000 is the appearance of the drum. We can also find that others background noise have effect of our calculate but not serious.

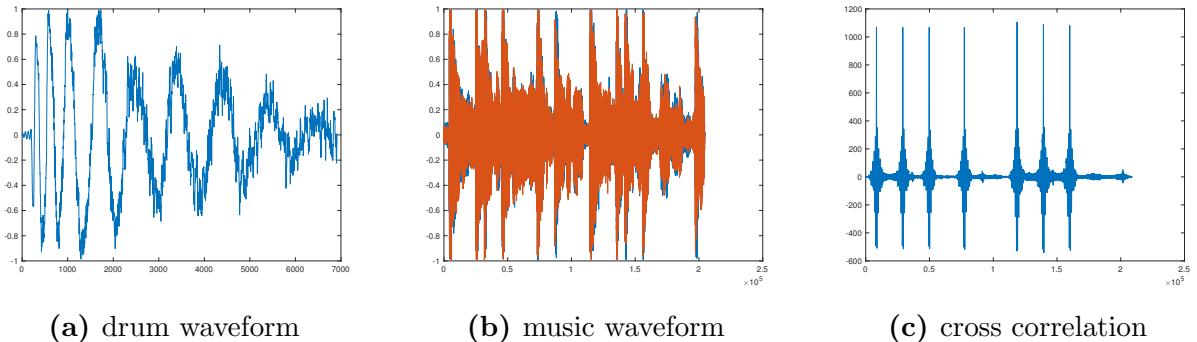


Figure 2.5: Two oscillographs

From the above plots we find one single drum's signal length is 7000, then we cut a 4.5 seconds music, length of it in Matlab is about $2 * 10^5$. And find 7 occurrences of the drum.

Source code for pattern finder is in Appendix A.1.7.

2.7 Cross correlation in three dimensions

This part is just some thinking by myself, not really theme related.

From the above research, we know that 1d cross correlation can be used to process acoustical signal and 2d cross correlation is very useful for image processing. Then, how about 3d cross correlation? If 3d cross correlation can be compute, we can easily know the space structure's similarity of two objects, theoretically. This task may have its value in microcosmos' research like molecular physics and microbiology. Consider we have one cell with particular disease as specimen and we do not know the reason of its pathogenicity. At the same time, we have several cells, some of which have the particular disease. If we can find spatial cross correlation between the specimen and other cell, base on the hypothesis that same diseased cell have the similar space structure, we can distinguish the healthy cell and diseased cell.

Computer vision is one of the most popular research today. One important reason is people believe 2d object is easy to understand and image compared with 3d object. The 2d technology today have been more and more mature. However, we live in a 3d (even higher dimensions) world, research in 3d will become mainstream someday, I believe 3d cross correlation will become a basic concept and tools like today's 2d cross correlation in the future.

Chapter 3

Calibration Model

In this part, we first explicit gaussian template build a dot detection model that can detect dots in calibration plate. We then build a calibration model use detected dots in several image pairs and their correspond coordinate in real space. After that, we employ the model to some image pairs and compute the depth of the scenery in the images. Meanwhile, several optimizatinos are proposed togher with the results of comparison experiments.

3.1 Dot Detection Algorithm

A common method used to calibrate a stereo vision system is to use a calibration plate. The computer vision system then needs to determine where the dots on the plate are. This can be achieved in many ways, but 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.

After computes cross-correlation of Gaussian template and calibration plate. We find all dots using local maximum method, that is, each time we find a peak in one part of the result matrix, and set the value in this area to 0. Untill no more peak can be find. Figure 3.1a shows the two dimensions gaussian template. We compute the cross correlation of the template with our calibration plate and finally detect the dots with ‘+’ mark as shown in Figure 3.1b.

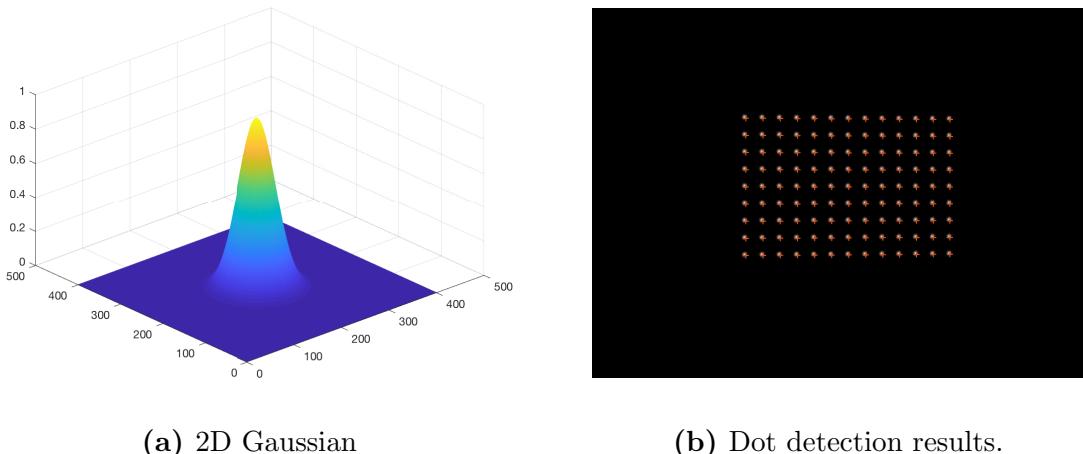
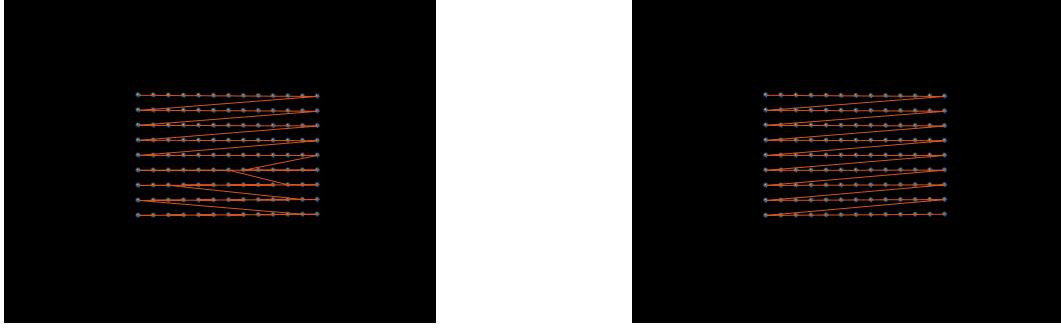


Figure 3.1: Dot dectection. The original dots are bright spots in the figure (b), detected dots are marked with red cross

The detected dots are saved in the order of its been detected, as shown in Figure 3.2a, the read line reveals the order of the dots been detect. However, we need a list of dots been sorted

from top to bottom, left to right. Sorted dots ishows in Figure 3.2b.



(a) Disordered dots

(b) Ordered dots

Figure 3.2: Sort dots from top to bottom, left to right.

Source code of Gaussian template matching, dot detection algorithm and dot sort are in Appendix A.2.1.

3.2 Create Calibration Model

The next step is to create a program which imports all the calibration images and records the pixel and real locations of all dots in each image. We use a fitting tool to create a 4D surface fit which connects all pixel space to real space within the calibrated zone. In this task, we have several calibration samples. Each sample contains two images, they are from a left and right camera, viewing the same calibration target at a stereo angle of approximately $\pm 9^\circ$. 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 and we know the distance of it to the cameras. So the dots' locations in real space is know and their locations in the left and right images can also be detected in task 1. Figure 3.3 demonstrate the mechanism of calibration model. In this figure, O and O' are two cameras and X is the object.

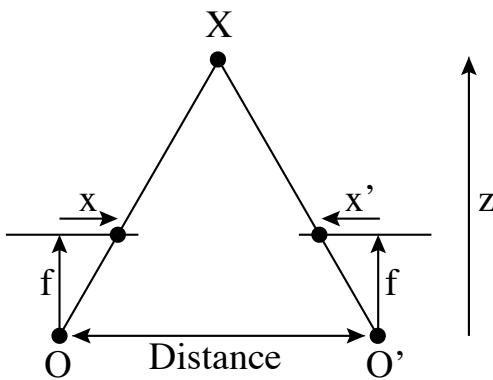


Figure 3.3: Calibration model machenism and extrinsic parameters calcuarion.

We use Matlab method `polyfitn` to build three fit functions for x , y and z coordinates in real space by using corresponding calibrate target's coordinates in left and right images, as

$$x_{real} = f_1(x_{left}, y_{left}, x_{right}, y_{right})$$

$$y_{real} = f_2(x_{left}, y_{left}, x_{right}, y_{right})$$

$$z_{real} = f_3(x_{left}, y_{left}, x_{right}, y_{right})$$

Besides, we can get more extrinsic parameters like camera's location and orientation in the world. In Figure 3.3 the distance between camera and object z is known, f is a camera parameter, x and x' are also known from two pictures. So we can calculate the distance between two cameras through similar triangle theorem.

Source code of calibration model creation is in Appendix A.2.2.

3.3 Image Comparision

After build calibration model. We hope to test the performance of our model use a pair of general images. But in practice, the images took by camera are not the dots with certain distribution rule. Preprocess of the image is needed before we apply it to our calibration model. Here, we first compare two images take by left and right camera, find corresponding objects in two images.

As observed in Chapter 1, the cross correlation technique is a very powerful tool which can sensitively and accurately identify patterns in complex images. Consider the two views of Physics South Building in Melbourne University, as shown in Figure 3.4 . As humans, most of us can easily recognize that both images are of the same scene, just taken from different angles or positions. But out computers do not have this ability and image comparison must be done to find corresponding objects in two images and then we can apply it to our calibration model.



Figure 3.4: Template created in left image (orange), search region created in right image (3 times larger than template, centered at the same pixel location).

We first break up one image in to windows and create a template from one window, like shown in Figure 3.4a , then create a search region (larger than the template) in the other image as shown in Figure 3.4b. Then we scan the template around the search region to find similar features using cross correlation, get the position of max correlation and compute the difference in pixel location, usually, we call this difference disparity. Repeat this for all windows and then we can get all corresponding positions which is very similar with the corresponding dots in task 1 and 2.

Image compare result shows in Figure 3.5. From which we find that most divided regions have a relatively high disparity. The reason may be that the images are taken by myself so that the distance between two cameras are long and they do not keep in a same high level.

For better demonstrate image comparison. We compare two particular images. Figure 3.6 shows the results, From which we can find the smaller windows size, the more information

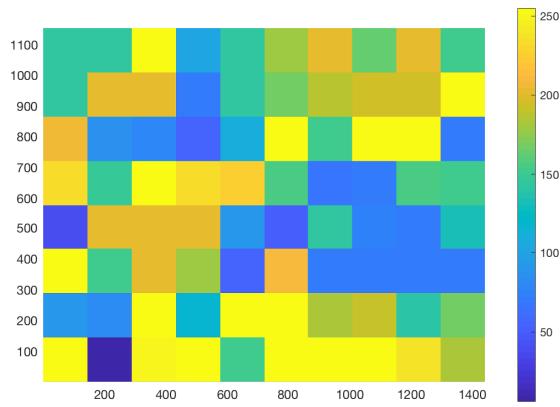


Figure 3.5: Image compare result, the region with color higher on the colorbar means the higher disparity

we can get. But because we set the search region is 3 times larger than the template, smaller windows may result in more templates that can not find a good match in the right view images. This problem will be solved in next task.

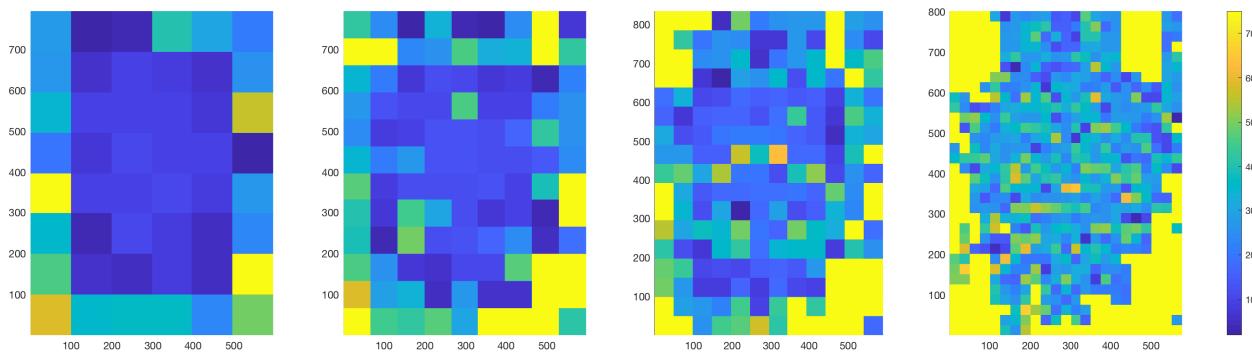


Figure 3.6: First two images are the image be compared, other four are results of different window size.

Source code about image comparison and result visualization are in Appendix A.2.3.

3.4 Cross Correlation Optimization

There are several parameters, which need to be considered when applying the cross correlation technique. These include what is the ideal sized window to be used, and how best should it be scanned in search of matching. In this part of the project, we investigate three optimization strategies:

a). Variable window overlap

The first optimization is variable window overlap. As mentioned in the last section, to do image compare, the left image will be split to several parts and each part will be searched in the corresponding but larger part of right images. Variable overlap is demonstrated in Figure 3.7, the split part could have overlap. The most important benefit is it increases the precision of disparity in one particular region, for example, one region will be compared only once if there is no overlap and it will be compared 4 times if there are 50% overlap, like the centre square of the right bottom part in Figure 3.7

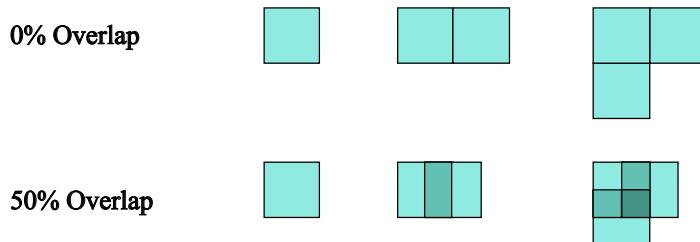


Figure 3.7: Example of variable window overlap

b). Variable search region geometry

Variable search region geometry, shown in Figure 3.8, is essential for a practical image compare model. On the one hand, if the search region is too small, the pattern may completely fall out of it and we can not get any useful difference information. However, if search region is too big, for example, treat the complete right view image as search region, it is fairly time consumed. So a proper size of search region is essential. On the other hand, a good shape of search region is also helpful, if our image is taken in the same horizontal level, we can use flat search region rather than square region, of course the search region should extend in horizontal rather than vertical, smaller search region could save time for us.

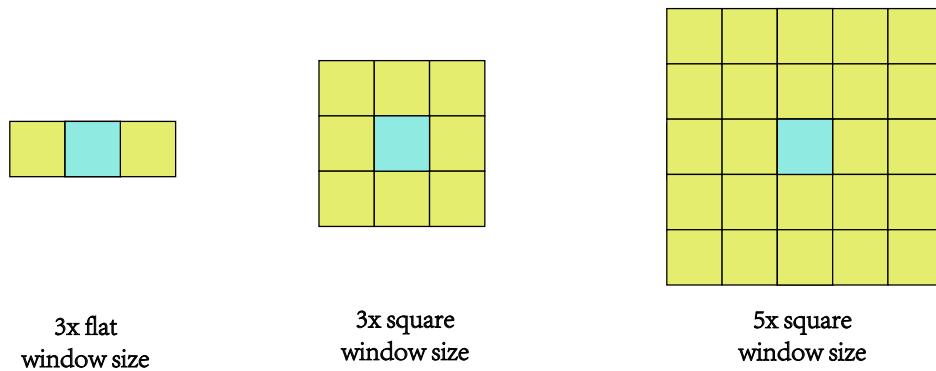


Figure 3.8: Example of variable search region

Actually, what image comparison does is that for each search region in the first image, find corresponding epipolar line in the right image, and examine all pixels on the epipolar line and pick the best match. In this project, stereo cameras are parallel to each other and camera

centers are at same height. Therefore, epipolar lines are fall along the horizontal scan lines of the images. Figure 3.9b shows the scan line and more reasonable search region.

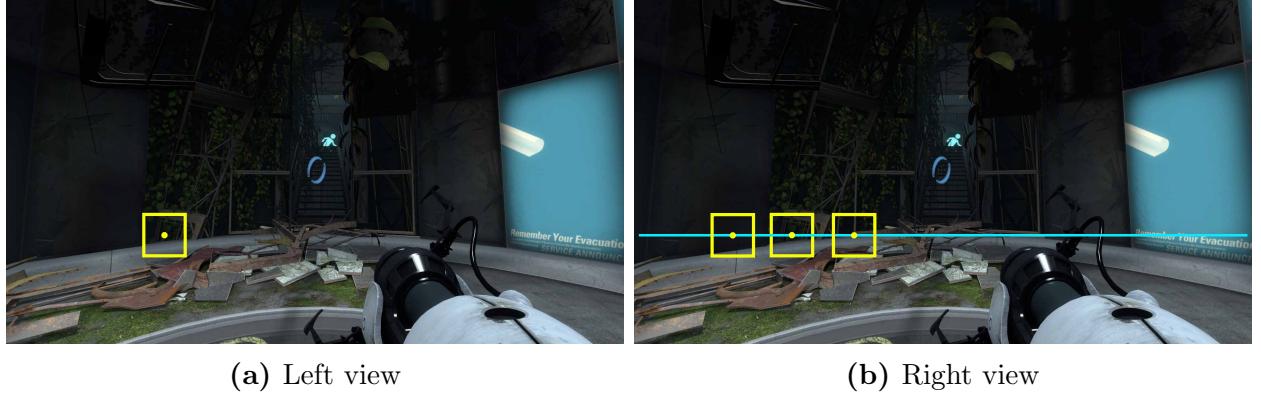


Figure 3.9: Scanlines and more reasonable search region

Source code of variable window overlap and variable search region geometry are in Appendix A.2.4.

c). Multi-Pass Cross Correlation

This process involves doing a course- to-fine multi-stage correlation. Consider the 2-pass cross correlation shown in Figure 3.10 as an example. In simple terms, the first pass is a broad guess at where the object has moved to and the second pass takes the information from this guess and provides finer detail.

For the first pass, the search region in the right image is centred at the same location as the template in the left image. This returns a pair (dpx , dpy) which is used to estimate center for the second pass.

In the second pass, the template was split into 4 smaller templates and search region in the right image is centred at the location of the template plus (dpx , dpy) and it returns 4 new centers for next pass.

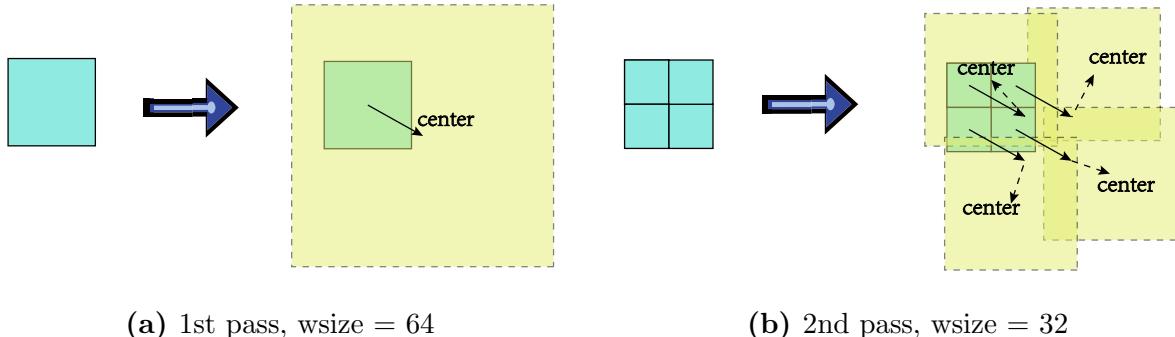


Figure 3.10: Multi-pass cross correlation. (“center” in the image refer to the new search region’s center, yellow square represents new search regions.)

However, Multi-pass has its flaw, if first pass compute a bad shift distance, the second pass will be influenced by this and tend to compute a worse shift distance.

Source code of multi-pass cross correlation optimization is in Appendix A.2.5.

3.5 Test Scan on Computer Generated Calibrated Images

In this section, we create 3D reconstruction of some test image pairs use fit functions and calibration model. Figure 3.11 shows the 3D reconstruction on three test image pairs. for each row in Figure 3.11, first two images are computer generated calibrated images from two views. The middle one is the image comparison result, the forth image is reconstructed objects in 3D space and last one is a clearer visualization of it.

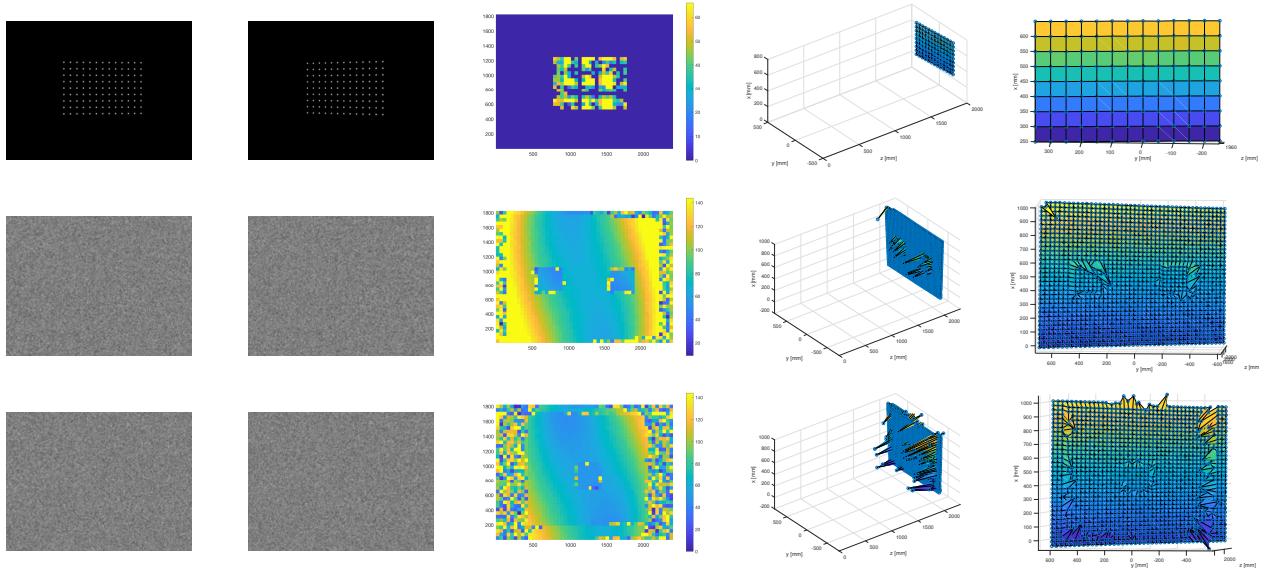


Figure 3.11: Test scan on computer generated calibrated images

From the above images we find that the image comparison does not work well at the edge of images of the second and third test piras. So we remove these vectors to get a more reasonable 3D reconstruction although there are still spurious exist.

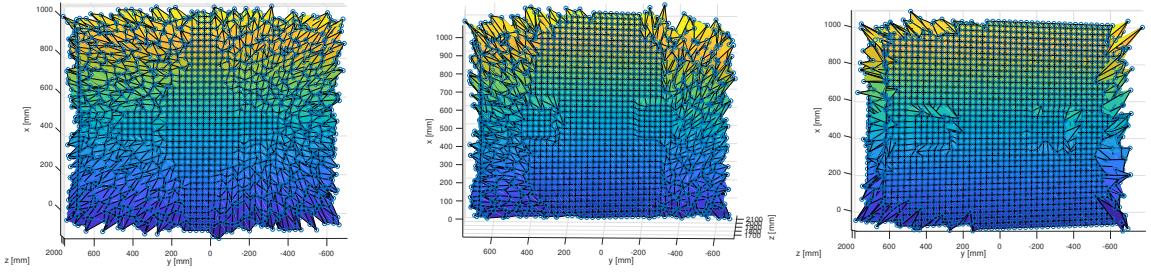
Source code for 3D restriction is in Appendix A.2.6 and A.2.7. There are many limitations of the program. First, we implement two versions code for test 1, and others. For test1, we use dot detextion algorithm which return all dots' position in two images. For test 2 and 3, we use image comparison algorithm but it dose not work well at the edge of the images.

3.6 Optimized Test Scan

In this section, we investage some optimization of test scan, Actually, the plots shows in last section are optimized test scan. Figure 3.12 show the progress of search region's optimizations. Figure 3.12a is the result of 1.5x flat search region and Figure 3.12b is the result of 1.5x square region. We can find results on square search region performs better than results on flat search region. Figure 3.12c is the result of 3x square search region, we find it is better than the result of 1.5x square search region. It may beacuse the larger search region could return a more precise results, of course it costs more time.

3.7 Better Calibration model

In this chapter, we build a calibration model and combina it with image comparison and finally reconstruct a 3D objects of images. There are some thinking of myself.



(a) 1.5x flat search region (b) 1.5x square search region (c) 3x square search region

Figure 3.12: Optimization of scan test

The built calibration model is base on the function fitting of several calibration plates. Although these plates distribute in different place in 3D space. They cover a little space of viewable range. So it works well only if the reconstructed objects in this little area. In this project, maybe $x \in (0, 1000)$, $y \in (-500, 500)$ and $z \in (1900, 2000)$. However, in our daily life, we want the depth information from some pictures took by camera, and the distance of the scenery may vary form 0 to very large number. The model proposed in the project may fail to this general task.

In my opinion, there are some solutions. The first one is use 3D calibration objects, actually, this idea is similar with using several 2D calibration plates with various z value. But use several 3D calibration objects may get more accurate calibration because it provides continuous fit sample for functions. Actually, maybe this method does not be mentioned because of the coding difficulty. The second approach is use several cameras, They do not have to be put in the same horizontal level, in contrary, they can be put in the four conors of a rectangle. Multi cameras give both robustness and precision of the model. Of course there should be better solutions, these two are just some thinking of my.

Chapter 4

Depth Map Prediction using Machine Learning

Thinking about the original inspiration of our project, human can easily estimate the distance between the scenery and themselves, we believe that is based on binocular vision. However, after thinking over, we may find we are not so sure about this hypothesis, because when we close one eye, we can still see the world and we can still estimate the depth of the scenery, maybe a little harder.

On the other hand, we did plenty of calculations in our projects, including build calibration model, image compare, to get the depth information. But do we really do this kind of calculations in our brain all the time? If the answer is yes, when do we begin to learn this approach? Do we estimate depth information simple based on the two images of left and right view? We may find the answer is no, because so many other cues are used when we estimate the distance of one object. For example, we know that one building should be far larger than a person and when a building and a person project to our retina with the same size, we should know the building is far and the person is near to us.

Actually, our brain does so many amazing things, the brain can learn to process images, sounds, it can learn to process our sense of touch. It seems like if we want to mimic the brain, we have to write lots of different pieces of software and programs. But instead, the way the brain does it is just worth a single learning algorithms, this is a hypothesis and widely accepted by scientists now. This magical algorithm is neural network algorithm.

Machine learning is a field of computer science that often uses statistical techniques to give computers the ability to "learn". In this chapter, we want to make a machine learn to predict the depth information using neural network, which is a machine learning task.

4.1 Convolutional Neural Network

Convolutional neural networks (CNNs) are the current state-of-the-art model architecture for image classification tasks. CNNs apply a series of filters to the raw pixel data of an image to extract and learn higher-level features, which the model can then use for classification and other image processing tasks.

In this section, we build a model to classify the images of hand written numbers, as shown in Figure 4.1. Through studying amount of features, the classifier can label a new image with the number on it, the accuracy can reach 0.99.

Usually, CNNs contains three components:

- Convolutional layers, which apply a specified number of convolution filters to the image. For each subregion, the layer performs a set of mathematical operations to produce a single value in the output feature map.

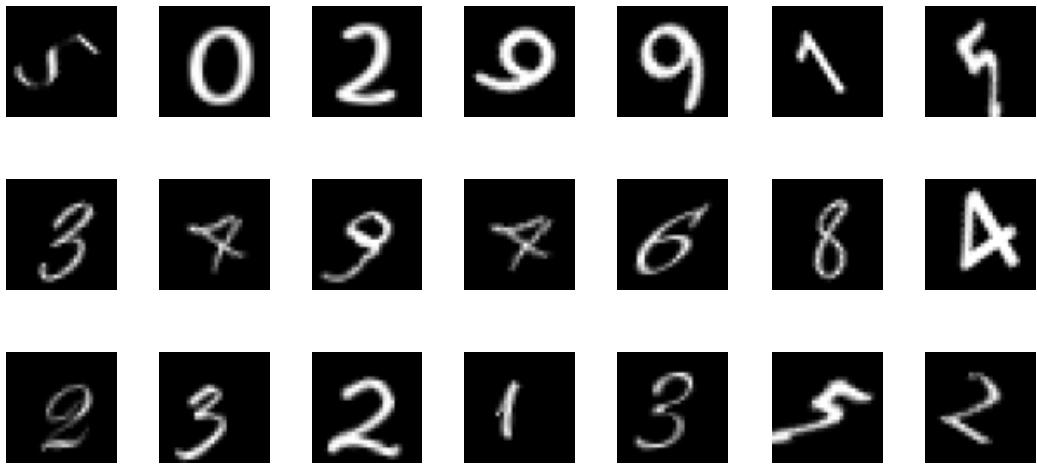


Figure 4.1: Hand write numbers

- Pooling layers, which downsample the image data extracted by the convolutional layers to reduce the dimensionality of the feature map in order to decrease processing time. A commonly used pooling algorithm is max pooling, which extracts subregions of the feature map (e.g., 2x2-pixel tiles), keeps their maximum value, and discards all other values. In our model, we use
- Dense (fully connected) layers, which perform classification on the features extracted by the convolutional layers and downsampled by the pooling layers. In a dense layer, every node in the layer is connected to every node in the preceding layer.

In our model, we define a deep neural network which contains one input layer, three convolution layers, two pooling layers and a output layer. The result shows deep networks performs better than shallow networks. Figure 4.2 shows the training progress of our model. From the figure we can easily find the accuracy of classify become higher and higher as the training epoch increase.

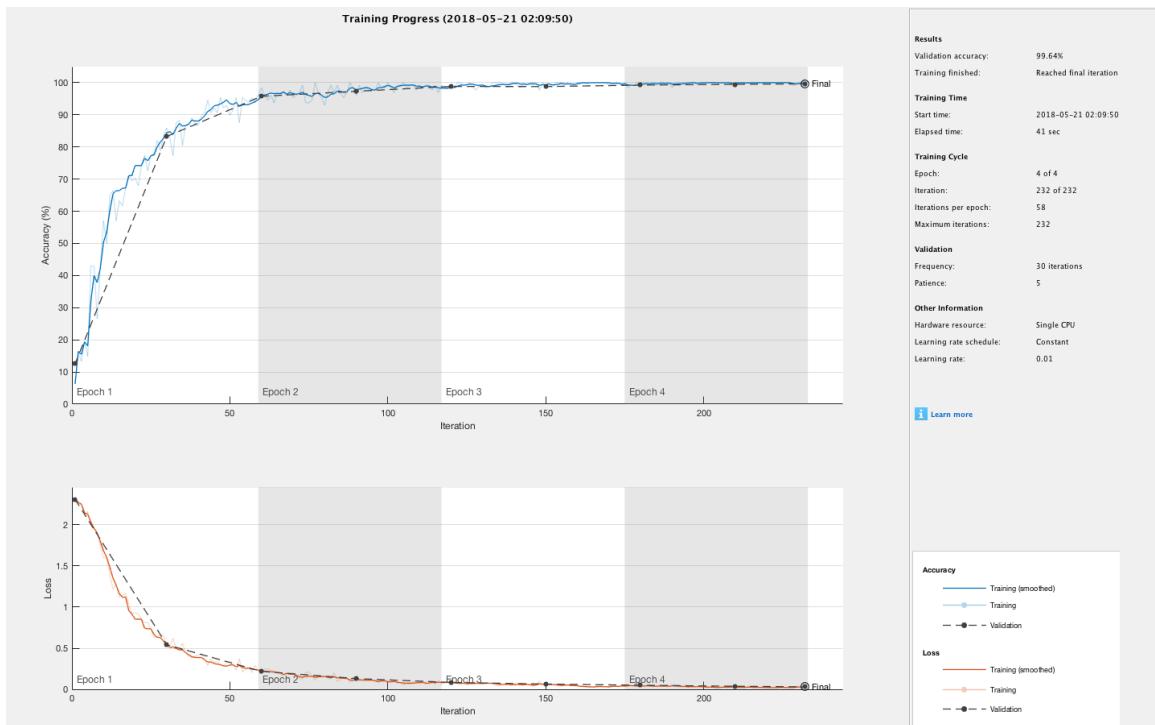


Figure 4.2: Training process

4.2 Stereo ConvNet

In this section, we introduce a stereo convolution neural network. The network is fully convolutional, and takes a couple of grayscale stereoscopic images concatenated along the channel axis, and outputs a single image representing the depth map. A series of convolutional and max-pooling layers followed by a series of upscaling and deconvolutional layers allow the network to extract image disparity features at the smaller scale (object edges), and generate a smooth estimate of the depth map at the larger scale (full object).

Figure 4.3 show the depth map prediction of Stereo ConvNet. The training/validation sets are created using the random virtual 3d scene generator *.

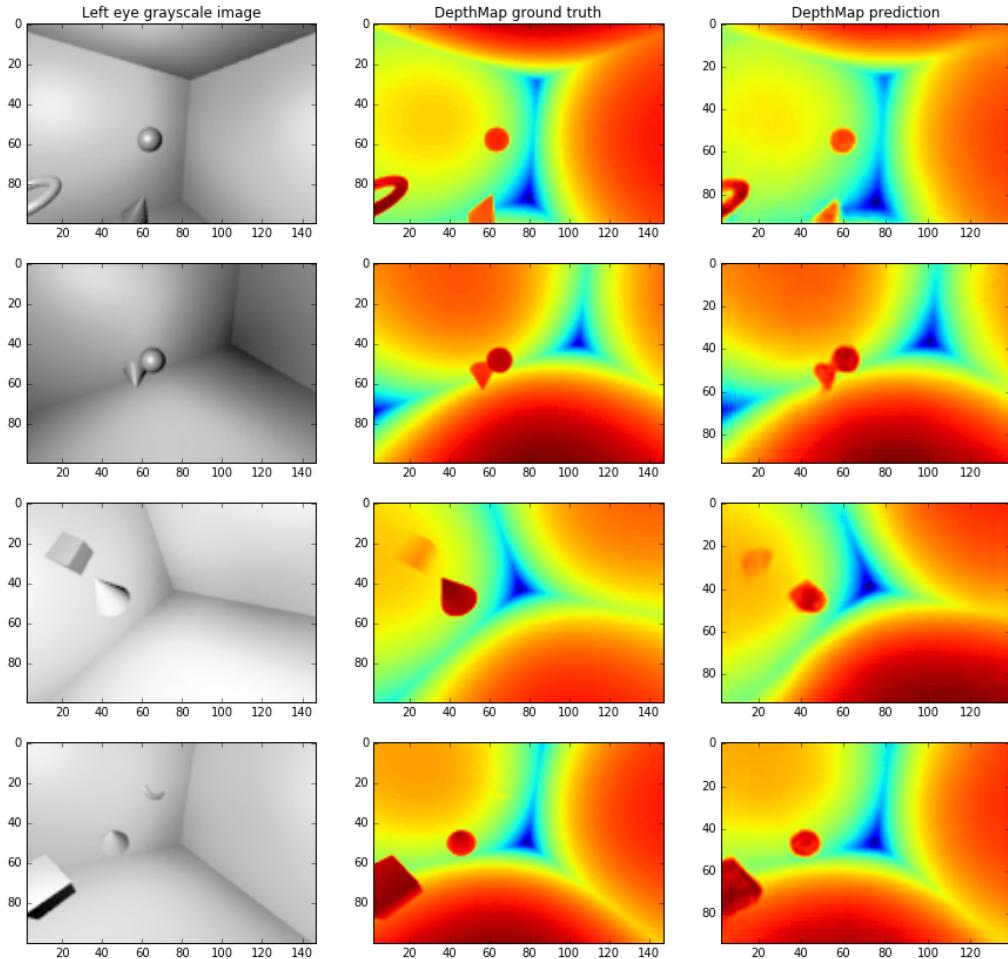


Figure 4.3: Stereo ConvNet results

*https://github.com/LouisFoucard/DepthMap_dataset

Appendix A

Matlab code

A.1 Part 1

A.1.1 Spatial Cross Correlation in 1d

```
1 %% %%%%%%%%
2 % Author: Haonan Li %
3 % Purpose: Compute the spatial cross correlation of two vectors %
4 %           with the same size %
5 %%%%%%%%
6
7 function vec_res = spatial_correlation_1d(vec_a,vec_b)
8 %% padding 0 in the head and tail of vec_a
9 len = length(vec_a);
10 vec_a = vec_a - mean(vec_a);
11 vec_b = [zeros(1, len-1), vec_b - mean(vec_b), zeros(1, len-1)];
12
13 %% cross correlation
14 vec_res = zeros(1, 2*len-1);
15 for i = 1:(2*len-1)
16     vec_res(i) = sum(vec_a .* (vec_b(1,i:i+len-1)))/len;
17 end
```

A.1.2 Normalised Spatial Cross Correlation in 1d

```
1 %%%%%%%
2 % Author: Haonan Li %
3 % Purpose: Compute the normalised cross correlation of two vectors %
4 %           of the same size %
5 %%%%%%%
6
7 function vec_res = normalised_spatial_correlation_1d(vec_a, vec_b)
8 %% normalization
9 len = length(vec_a);
10 vec_a = vec_a - mean(vec_a);
11 vec_b = vec_b - mean(vec_b);
12 %% compute sigma
13 sigma = sqrt(sum(vec_a.^2)*sum(vec_b.^2))
14 %% padding 0 in the head and tail of vec_b
15 vec_b = [zeros(1, len-1), vec_b, zeros(1, len-1)];
```

```

16
17 %% cross correlation
18 vec_res = zeros(1, 2*len-1);
19 for i = 1:(2*len-1)
20     % nor_a, nor_b are the normalization factors
21     vec_res(i) = sum(vec_a .* (vec_b(1,i:i+len-1)))/sigma;
22 end

```

A.1.3 Signal Offset

```

1 %%%%%%
2 % Author: Haonan Li %
3 % Purpose: Give two signal file with signals come from the same %
4 % source and just offset by some time, find the offset %
5 % time and sensor distance %
6 %%%%%%
7
8 function offset = signal_offset_checker(f_1, f_2)
9 %% read data from file
10 file_1 = importdata(f_1);
11 file_2 = importdata(f_2);
12 sig_1 = file_1.data';
13 sig_2 = file_2.data';
14 SAMPLE_RATE = 44100;
15
16 %% compute cross correlation of two signal, log run time
17 tic;
18 % special method
19 cross_cor = spatial_correlation_1d(sig_1, sig_2)';
20 % spectral method (faster)
21 % cross_cor = spectral_correlation_function(sig_1, sig_2)';
22 run_time = toc
23
24 %% find the position of max cross coorelation value and compute the offset.
25 [max_value, max_pos] = max(abs(cross_cor));
26 offset = abs(length(sig_1) - max_pos)
27
28 % compute offset time and sensor distance
29 offset_time = offset / SAMPLE_RATE
30 distance = 333 * offset_time

```

A.1.4 Normalised Spatial Cross Correlation in 2d

```

1 %%%%%%
2 % Author: Haonan Li %
3 % Purpose: Receive two matrix, t(template) and A(search region) %
4 % return normalized cross-correlation of these two matrix %
5 %%%%%%
6
7 function mat_res = normalised_spatial_correlation_2d(mat_t, mat_A)
8 %% init mat_res with 0
9 [tY,tX] = size(mat_t);
10 [AY,AX] = size(mat_A);
11 mat_res = zeros(AY-tY+1, AX-tX+1);

```

```

12 [rY,rX] = size(mat_res);
13
14 %% cross correlation
15 % mat_t_ is a matrix of mat_t minus mean(mat_t)
16 mat_t_ = mat_t - mean(mat_t);
17 for i = 1:rY
18     for j = 1:rX
19         % compute mat_res(i,j)
20         % first get the part of mat_A covered by the mat_t
21         mat_A_under = mat_A(i:(i+tY-1), j:(j+tX-1));
22         mat_A_under_ = mat_A_under - mean(mat_A_under);
23         % compute numerator and denominator of the cross correlation
24         numerator = sum(sum(mat_t_ .* mat_A_under_));
25         denominator = sqrt(sum(sum(mat_t_.^2)) * sum(sum(mat_A_under_.^2)));
26         mat_res(i,j) = numerator/denominator;
27     end
28 end
29
30 %% figure
31 figure,
32 surf(mat_res);
33 figure(gcf)

```

A.1.5 Image Alignment

```

1 %%%%%%
2 % Author: Haonan Li %
3 % Purpose: Give two pictures, one is a part of another find the %
4 %           corresponding position and mark it %
5 %%%%%%
6
7 function [pos_y, pos_x] = find_the_rocket_man(pic_t, pic_A)
8 %% read images
9 pic_t = imread(pic_t);
10 pic_A = imread(pic_A);
11 mat_t = mean(pic_t, 3);
12 mat_A = mean(pic_A, 3);
13 [tY,tX] = size(mat_t);
14
15 %% compute cross correlation
16 tic;
17 cross_corr = normalised_spatial_correlation_2d(mat_t,mat_A);
18 % cross_corr = my_norm_xcorr2_2(mat_A,mat_t);
19 run_time = toc
20
21 % max cross correlation position
22 [pos_y, pos_x] = find(cross_corr == max(max(cross_corr)));
23
24 %% save marked image
25 imshow(pic_A);
26 hold on;
27 marker = plot(pic_A(1,1), 'p');
28 marker.XData = pos_x + tX/2;
29 marker.YData = pos_y + tY/2;
30 marker.MarkerSize = 30;
31 marker.Color = 'r';
32 marker.MarkerFaceColor = 'r';

```

```
33 saveas(gcf, 're','png');
```

A.1.6 Spectral Cross Correlation

```
1 %%%%%%
2 % Author: Haonan Li %
3 % Purpose: Compute the spectral cross correlation of two vectors %
4 % of the same size using Fourier transform. %
5 %%%%%%
6
7 function vec_res = spectral_correlation_1d(vec_a, vec_b)
8
9 f_a = fft(vec_a);
10 f_b = fft(vec_b);
11 conj_f_a = conj(f_a);
12 mid_res = conj_f_a .* f_b;
13 vec_res = ifft(mid_res);
14
15
16 %% My fft, does not work if NN is not the power of 2
17 function res_fft = my_fft(vec)
18
19 vec_size = size(vec);
20 N = vec_size(2)
21 c = zeros(1,N);
22 % indexing computation
23 j1 = 0;
24 for i = 1 : N
25     if i < j1 + 1
26         tmp = vec(j1 + 1);
27         vec(j1 + 1) = vec(i);
28         vec(i) = tmp;
29     end
30     k = N / 2;
31     while k ≤ j1
32         j1 = j1 - k;
33         k = k / 2;
34     end
35     j1 = j1 + k;
36 end
37
38 %% Butterfly computing
39 dig = 0;
40 k = N;
41 while k > 1
42     dig = dig + 1;
43     k = k / 2;
44 end
45
46 n = N / 2;
47 for m = 1 : dig
48     dist = 2 ^ (m - 1);
49     idx = 1;
50     for i = 1 : n
51         idx1 = idx;
52         for j1 = 1 : N / (2 * n)
53             r = (idx - 1) * 2 ^ (dig - m);
```

```

54         coef = exp(j * (-2 * pi * r / N));
55         tmp
56             = vec(idx);
57         vec(idx)      = tmp + vec(idx + dist) * coef;
58         vec(idx + dist) = tmp - vec(idx + dist) * coef;
59         idx = idx + 1;
60     end
61     idx = idx1 + 2 * dist;
62 end
63 n = n / 2;
64 res_fft = vec;

```

A.1.7 Pattern Finder

```

1 %%%%%%
2 % Author: Haonan Li %
3 % Purpose: Find all assurance of a particular element in a song %
4 %%%%%%
5
6 function res = pattern_finder(wavfile, patfile)
7 %% load a music and a particular element (drum)
8 [y1,Fs1] = audioread(wavfile);
9 x1 = y1(:,1);
10 [y2,Fs2] = audioread(patfile);
11 x2 = y2(:,1);
12
13 %% cross correlation
14 corr_vec = xcorr(x1',x2');
15 res = corr_vec;
16
17 %% show
18 % plot(y1);
19 % plot(y2);
20 plot(res);

```

A.2 Part 2

A.2.1 Dot Detection Algorithm

```

1 %%%%%%
2 % Author: Haonan Li %
3 % Purpose: Detect dots on a calibration plate %
4 %%%%%%
5
6 function dots = dot_detect(img_t)
7 %% read image
8 img = imread(img_t);
9 mat_img = rgb2gray(img);
10 [X,Y] = size(mat_img);
11
12 %% create a Gaussian template
13 u = [-2:2];
14 v = [-2:2];

```

```

15 [U,V] = meshgrid(u,v);
16 mat_gauss = exp(-(U.^2+V.^2)./2/3^2);
17 % draw the gaussian template
18 % u = [-20:0.1:20];
19 % v = [-20:0.1:20];
20 % [U,V] = meshgrid(u,v);
21 % mat_gauss = exp(-(U.^2+V.^2)./2/3^2);
22 % surf(mat_gauss), shading flat
23 % hold on
24
25 %% cross correlation
26 cross_corr = xcorr2(mat_img, mat_gauss);
27 % local max cross correlation position
28 dots = local_max(cross_corr,50);
29 % dots = order_dots(dots,21,17);
30 dots = order_dots(dots,13,9);
31
32 % draw the point in the picture
33 % imshow(img)
34 % hold on
35 % plot(dots(:,2),dots(:,1),'+', 'Markersize',10)
36 % % draw line to check the point order
37 % plot(dots(:,2),dots(:,1))
38 end
39
40
41 %% find all local maximum dots, level is the local range
42 function res = local_max(mat_a, level)
43 res = [];
44 [X,Y] = size(mat_a);
45 x = 1+level;
46 while x < X-level
47     y = 1+level;
48     while y < Y-level
49         if mat_a(x,y) > 0
50             local_maxx = max(max(mat_a(x-level:x+level,y-level:y+level)));
51             if mat_a(x,y) == local_maxx
52                 res = [res;x,y];
53                 % if find one local max, set positions around it to 0
54                 mat_a(x-level:x+level,y-level:y+level) = 0;
55             end
56         end
57         y = y + 1;
58     end
59     x = x + 1;
60 end
61 end
62
63 %% order the dots
64 function res = order_dots(dots, n, m)
65 res = dots;
66 for i=1:m
67     for j=1:n
68         st = (i-1)*n+1;
69         en = i*n;
70         res(st:en,:) = sortrows(dots(st:en,:),2);
71     end
72 end
73 end

```

A.2.2 Create Calibration Model

```
1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % Author: Haonan Li %
3 % Purpose: Create a calibration model use a set of calibration %
4 % images %
5 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6
7 function [fitfunx, fitfuny, fitfunz] = calibration_model()
8 %% input all calibration images
9 real_dot = [];
10 left_dot = [];
11 right_dot = [];
12 for i = 0:5
13     dis_z = 1900 + i * 20;
14     left_img = ['Resources/cal_image_left_', num2str(dis_z), '.tiff'];
15     right_img = ['Resources/cal_image_right_', num2str(dis_z), '.tiff'];
16     real_dot = [real_dot; build_dot(dis_z)];
17     left_dot = [left_dot; dot_detect(left_img)];
18     right_dot = [right_dot; dot_detect(right_img)];
19 end
20
21 %% build fit dunctions
22 ly = left_dot(:,1);
23 lx = left_dot(:,2);
24 ry = right_dot(:,1);
25 rx = right_dot(:,2);
26 realy = real_dot(:,1);
27 realx = real_dot(:,2);
28 realz = real_dot(:,3);
29 capture = [lx,ly,rx,ry];
30
31 fitfunx = myfun(capture, realx);
32 fitfuny = myfun(capture, realy);
33 fitfunz = myfun(capture, realz);
34 end
35
36 %% compute dots positions in real space
37 function real = build_dot(real_z)
38 for j = 1:17
39     for i = 1:21
40         index = 21*(j-1)+i;
41         real(index,:) = [j*50,-500+i*50,real_z];
42     end
43 end
44 end
45
46 %% output the calibration model
47 function func = myfun(capture, real)
48 fitfun = polyfitn(capture, real, 3);
49 sx      = (polyn2sym(fitfun));
50 func   = matlabFunction(sx);
51 end
```

A.2.3 Image Comparision

```

1 %%%%%%%%%%%%%%%
2 % Author: Haonan Li %
3 % Purpose: Compare two images, split one image to several windows, %
4 %           search more similar part in the other corresponding %
5 %           image %
6 %%%%%%%%%%%%%%
7
8 function res = image_compare(img_a, img_b)
9 %% input images
10 img_a = imread(img_a);
11 A = rgb2gray(img_a);
12 img_b = imread(img_b);
13 B = rgb2gray(img_b);
14 [Y,X] = size(B);
15
16 %% image comparasion
17 % store the result
18 diff = A.*0;
19 res = [];
20 % split the larger side length to 10
21 wsize = round(max(X, Y) / 10);
22 for i = 0 : ceil(X/wsize) - 1
23     for j = 0 : ceil(Y/wsize) - 1
24         xgrid = 1 + i * wsize;
25         ygrid = 1 + j * wsize;
26         % find correspond points of two images
27         [dx,dy] = corr_search(A, B, wsize, xgrid, ygrid);
28         res = [res;xgrid,ygrid,xgrid+dx,ygrid+dy];
29         diff(ygrid:(ygrid+wsize),xgrid:(xgrid+wsize)) = sqrt(dx*dx+dy*dy);
30     end
31 end
32
33 %% show
34 ssize = 10*wsize;
35 diff(diff>ssize) = ssize;
36 surf(diff)
37 axis image
38 shading interp
39 colorbar
40 hold on
41 end
42
43
44 function [dx,dy] = corr_search(mat_a, mat_b, wsize, xgrid, ygrid)
45 %% pattern (window)
46 [Y,X] = size(mat_a);
47 pat_left = xgrid;
48 pat_right = min(X,xgrid + wsize - 1);
49 pat_top = ygrid;
50 pat_bottom = min(Y,ygrid + wsize - 1);
51 pattern = mat_a(pat_top:pat_bottom, pat_left:pat_right);
52
53 %% search region: 3x window size square
54 sr_left = max(1, pat_left - wsize + 1);
55 sr_right = min(X, pat_right + wsize);
56 sr_top = max(1, pat_top - wsize + 1);
57 sr_bottom = min(Y, pat_bottom + wsize);
58 search_region = mat_b(sr_top:sr_bottom, sr_left:sr_right);
59
60 %% find the max cross correlation position

```

```

61 cross_corr = my_norm_xcorr2(search_region, pattern);
62 [rel_y,rel_x] = find(cross_corr == max(cross_corr(:)));
63 if isempty(rel_x)
64     dx = 1000;
65     dy = 1000;
66 else
67     dx = rel_x(1) + sr_left - pat_left-1;
68     dy = rel_y(1) + sr_top - pat_top-1;
69 end
70 end
71
72 %% 2d cross correlation
73 function mat_r = my_norm_xcorr2(mat_A, mat_t)
74 [Ay, Ax] = size(mat_A);
75 [ty,tx] = size(mat_t);
76 % Change - Compute the cross power spectrum
77 Ga = fft2(mat_A);
78 Gb = fft2(mat_t, Ay, Ax);
79 mat_r = real(ifft2((Ga.*conj(Gb))./abs(Ga.*conj(Gb))));
```

A.2.4 Cross Correlation Optimisation

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % Author: Haonan Li %
3 % Purpose: Optimization of image compare, overlap refers window %
4 %           overlap. Range in [0,1), sr_size is the what times %
5 %           search size of window, search shape values [s]: square %
6 %           or [f]: flat. %
7 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
8
9 function res = image_compare_optimized(img_a, img_b, overlap, sr_size, ...
    sr_shape)
10 %% input images
11 img_a = imread(img_a);
12 A = rgb2gray(img_a);
13 img_b = imread(img_b);
14 B = rgb2gray(img_b);
15 [Y,X] = size(B);
16
17 %% image comparation
18 res = [];
19 diff = A.*0;
20 % decide window size
21 wsize = round(max(X, Y) / 50 );
22 % compute shift size by overlap
23 shift = ceil(wsize * (1 - overlap));
24 ceil((X-wsize)/shift)
25 ceil((Y-wsize)/shift)
26 for i = 0 : ceil((X-wsize)/shift)
27     for j = 0 : ceil((Y-wsize)/shift)
28         xgrid = 1 + i * shift;
29         ygrid = 1 + j * shift;
30         % find correspond points of two images
31         [dx,dy] = corr_search(A, B, wsize, xgrid, ygrid, sr_size, sr_shape);
32         res = [res;xgrid,ygrid,xgrid+dx,ygrid+dy];
33         diff(ygrid:(ygrid+wsize),xgrid:(xgrid+wsize)) = sqrt(dx*dx+dy*dy);
```

```

34     end
35 end
36
37 %% draw the corresponding points
38 diff(diff>3*wsize) = 3*wsize;
39 surf(diff)
40 axis image
41 shading interp
42 colorbar
43 hold on
44 end
45
46
47 %% optimized compare, two options (overlap and search region shape)
48 function [dx,dy] = corr_search(A, B, wsize, xgrid, ygrid, sr_size, sr_shape)
49 %% pattern (window)
50 [Y,X] = size(B);
51 pat_left = xgrid;
52 pat_right = min(X,xgrid + wsize - 1);
53 pat_top = ygrid;
54 pat_bottom = min(Y,ygrid + wsize - 1);
55 pattern = A(pat_top:pat_bottom, pat_left:pat_right);
56
57 %% search region
58 extend = floor(wsize * (sr_size - 1)/2);
59 sr_left = max(1, pat_left - extend + 1);
60 sr_right = min(X, pat_right + extend);
61 % search region shape
62 if sr_shape == 's' % square
63     sr_top = max(1, pat_top - extend + 1);
64     sr_bottom = min(Y, pat_bottom + extend);
65 elseif sr_shape == 'f' % flat
66     sr_top = pat_top;
67     sr_bottom = pat_bottom;
68 end
69 search_region = B(sr_top:sr_bottom, sr_left:sr_right);
70
71 %% find the max cross correlation position
72 cross_corr = normxcorr2(pattern, search_region);
73 [rel_y,rel_x] = find(cross_corr == max(cross_corr(:)));
74 if isempty(rel_x)
75     dx = 0;
76     dy = 0;
77 else
78     dx = rel_x(1) + sr_left - pat_left-1;
79     dy = rel_y(1) + sr_top - pat_top-1;
80 end
81 end
82
83 %% 2d cross correlation
84 function mat_r = my_norm_xcorr2(mat_A, mat_t)
85 [Ay, Ax] = size(mat_A);
86 [ty,tx] = size(mat_t);
87 % Change - Compute the cross power spectrum
88 Ga = fft2(mat_A);
89 Gb = fft2(mat_t, Ay, Ax);
90 mat_r = real(ifft2((Ga.*conj(Gb))./abs(Ga.*conj(Gb))));
```

A.2.5 Multi-pass Cross Correlation

```

1 %%%%%%%%%%%%%%%%%
2 % Author: Haonan Li %
3 % Purpose: Multipass image comparation: recursively call image %
4 %           comparation function of part of the whole image %
5 %%%%%%%%%%%%%%%%
6
7 function diff = image_compare_multipass(img_a, img_b)
8 %% input images
9 img_a = imread(img_a);
10 A = rgb2gray(img_a);
11 img_b = imread(img_b);
12 B = rgb2gray(img_b);
13 [Y,X] = size(B);
14
15 %% image comparasion
16 % store the result
17 diff = A.*0;
18 % init window size
19 wsize = 128;
20 % recursive level
21 level = 2;
22 for i = 0 : ceil(X/wsize) - 1
23     for j = 0 : ceil(Y/wsize) - 1
24         xgrid = 1 + i * wsize;
25         ygrid = 1 + j * wsize;
26         % find correspond points of two images
27         diff = img_cmp(A, B, wsize, xgrid, ygrid, 0, 0, diff, 2);
28     end
29 end
30
31 %% show
32 ssize = 3*wsize/2^level;
33 diff(diff>ssize) = ssize;
34 surf(diff)
35 axis image
36 shading interp
37 colorbar
38 hold on
39 end
40
41 %% recursive compare function
42 function diff = img_cmp(A, B, wsize, xgrid, ygrid, dpx, dpy, diff, level)
43 %% pattern (window)
44 [Y,X] = size(A);
45 pat_left = xgrid;
46 pat_right = min(X,xgrid + wsize - 1);
47 pat_top = ygrid;
48 pat_bottom = min(Y,ygrid + wsize - 1);
49 pattern = A(pat_top:pat_bottom, pat_left:pat_right);
50
51 %% search region: 3x window size square
52 sr_left = max(1, pat_left - wsize + 1 + dpx);
53 sr_right = min(X, pat_right + wsize + dpx);
54 sr_top = max(1, pat_top - wsize + 1 + dpy);
55 sr_bottom = min(Y, pat_bottom + wsize + dpy);
56 % if search region out of bound completely, stop
57 if sr_left > X | sr_top > Y

```

```

58     diff(pat_top:pat_bottom, pat_left:pat_right) = sqrt(2000000);
59     return
60 end
61 search_region = B(sr_top:sr_bottom, sr_left:sr_right);
62
63 % find the max cross correlation position
64 cross_corr = my_norm_xcorr2(search_region, pattern);
65 [rel_y, rel_x] = find(cross_corr == max(cross_corr(:)));
66 % can not find a cross corr result in big picture
67 if isempty(rel_x) & level > 1
68     dx = 0;
69     dy = 0;
70 elseif isempty(rel_x) & level == 1
71     dx = 1000;
72     dy = 1000;
73 else
74     dx = rel_x(1) + sr_left - pat_left-1;
75     dy = rel_y(1) + sr_top - pat_top-1;
76 end
77 diff(pat_top:pat_bottom, pat_left:pat_right) = sqrt(dx*dx+dy*dy);
78 % if level == 1, stop recursive
79 if level == 1
80     return
81 else
82     wsize = wsize / 2;
83 end
84
85 %% recursively call image compare
86 level = level-1;
87 xgrid1 = pat_left;
88 xgrid2 = pat_left+wsize;
89 ygrid1 = pat_top;
90 ygrid2 = pat_top+wsize;
91 diff = img_cmp(A, B, wsize, xgrid1, ygrid1, dx, dy, diff, level);
92 if xgrid2 < X
93     diff = img_cmp(A, B, wsize, xgrid2, ygrid1, dx, dy, diff, level);
94 end
95 if ygrid2 < Y
96     diff = img_cmp(A, B, wsize, xgrid1, ygrid2, dx, dy, diff, level);
97 end
98 if xgrid2 < X & ygrid2 < Y
99     diff = img_cmp(A, B, wsize, xgrid2, ygrid2, dx, dy, diff, level);
100 end
101 end
102
103 %% 2d cross correlation
104 function mat_r = my_norm_xcorr2(mat_A, mat_t)
105 [Ay, Ax] = size(mat_A);
106 [ty, tx] = size(mat_t);
107 % Change - Compute the cross power spectrum
108 Ga = fft2(mat_A);
109 Gb = fft2(mat_t, Ay, Ax);
110 mat_r = real(ifft2((Ga.*conj(Gb))./abs(Ga.*conj(Gb))));
```

A.2.6 Multi-pass Cross Correlation

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Author: Haonan Li %
3  % Purpose: create a 3D reconstruction of the test image pairs %
4  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5
6  function res = test_scan1(img_a, img_b, fx, fy, fz)
7  %% compare two images
8  fprintf("Image comparing...\n");
9  left_dot = dot_detect(img_a);
10 right_dot = dot_detect(img_b);
11 cmp = [left_dot(:,2),left_dot(:,1),right_dot(:,2),right_dot(:,1)];
12 fprintf("Image comparing done!\n");
13
14 %% init calibrate model (it may take a long time, so we make it parameters)
15 fprintf("Calibration model initializing...\n");
16 % [fx, fy, fz] = calibration_model();
17 fprintf("Calibration model initialize done!\n");
18
19 %% create a 3D reconstruction
20 [X,Y] = size(cmp)
21 res = [];
22 for i = 1:X
23     realx = fx(cmp(i,1),cmp(i,2),cmp(i,3),cmp(i,4));
24     realy = fy(cmp(i,1),cmp(i,2),cmp(i,3),cmp(i,4));
25     realz = fz(cmp(i,1),cmp(i,2),cmp(i,3),cmp(i,4));
26     res = [res;cmp(i,:),realx,realy,realz];
27 end
28
29 %% Plot
30 figure;
31 plot3(res(:,7),res(:,5),res(:,6), 'o')
32 hold on
33 surf(reshape(res(:,7),[13,9]), reshape(res(:,5),[13,9]), reshape(res(:,6),[13,9]))
34 xlabel('z [mm]')
35 ylabel('y [mm]')
36 zlabel('x [mm]')
37 axis equal
38 axis([0,2000, -500,500, 0,800]);
39 grid on

```

A.2.7 Multi-pass Cross Correlation

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Author: Haonan Li %
3  % Purpose: create a 3D reconstruction of the test image pairs %
4  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5
6  function res = test_scan(img_a, img_b, fx, fy, fz)
7  %% compare two images
8  fprintf("Image comparing...\n");
9  cmp = image_compare_optimized(img_a,img_b,0,5,'s');
10 size(cmp)
11 fprintf("Image comparing done!\n");
12
13 %% init calibrate model (it may take a long time, so we make it parameters)
14 fprintf("Calibration model initializing...\n");
15 % [fx, fy, fz] = calibration_model();

```

```

16 fprintf("Calibration model initialize done!\n ");
17
18 %% create a 3D reconstruction
19 [X,Y] = size(cmp)
20 res = [];
21 for i = 1:X
22     realx = fx(cmp(i,1),cmp(i,2),cmp(i,3),cmp(i,4));
23     realy = fy(cmp(i,1),cmp(i,2),cmp(i,3),cmp(i,4));
24     realz = fz(cmp(i,1),cmp(i,2),cmp(i,3),cmp(i,4));
25     res = [res;cmp(i,:),realx,realy,realz];
26 end
27
28 %% Plot
29 figure;
30 plot3(res(:,7),res(:,5),res(:,6),'o')
31 hold on
32 fsize = [38,51];
33 surf(reshape(res(:,7),fsize),reshape(res(:,5),fsize),reshape(res(:,6),fsize))
34 xlabel('z [mm]')
35 ylabel('y [mm]')
36 zlabel('x [mm]')
37 axis equal
38 % axis([0,2300, -800,800, -200,1000]);
39 grid on

```

Bibliography