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

25 May 2018

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

Contents

Chapter 1

Introduction

As we all know, natural selection is the progress that eliminates inferior species gradually over time and organisms today are products of it. In other words, the biological structures of livings are fitting for current environment. In the mean time, It is clearly obvious that most animals have two eyes (referred to as Binocular vision). So what advantages of this biological structure? Why it is the choice of nature rather than single eye? One possible reason is that having two eyes provides a much wider field of view. However, research shows that humans have a maximum horizontal field of view of 194 degrees with two eyes but 114 degrees of this is the binocular field of view (seen by both eyes)[6]. 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, relative depth information can be extracted 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.

Chapter 2 is about implementing spatial cross correlation computation model. In the first two sections, one dimensional cross correlation model is built, which is very useful in signal processing. To test the effectiveness of it, we apply it to a particular signal offset compute task. In the next two sections, the model is extended to two dimensions, which can work in image comparison. Similarly, the model is applied to a particular pattern search task. Cause spatial cross correlation is time consuming, a faster method to achieve same computational ability of cross correlation is needed. So we explore and implement spectral cross correlation in section 5, and compare the efficiency of two methods. The next section is an application of one dimensional cross correlation about pattern detection.

In Chapter 3, the image comparison model and calibration model are built separately and combined to reconstruct a 3D scenery from two 2 calibrate images. In first two sections, calibration model is created by using calibration plants. In section 3, image comparison model is built and a mapping algorithm to extract depth data from comparison result is presented. Section 4 is about some optimization strategies to image comparison model, for speed or accuracy. Section 5 and 6 is the reconstruction of 3D scenery using models presented before.

Chapter 2

Cross Correlations

Cross correlation is powerful 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 calculating cross correlation. To be specific, large signals and images may result in time consumption. Then there is an alternative way to solve this problem, that is, finding spectral cross correlation with a Fourier transform and inverse Fourier transform. 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[15][16]:

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

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-quare. 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

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

Source code of one dimensional cross correlation and normalized one dimensional cross correlation are in Appendix A.1.1 andA.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 since they are captured by two sensors with constant distance. Cross correlation can be used to find the offset time and distance between two sensors. 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. Through the given sample rate and propagation speed, the offset time and distance can be calculated 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.

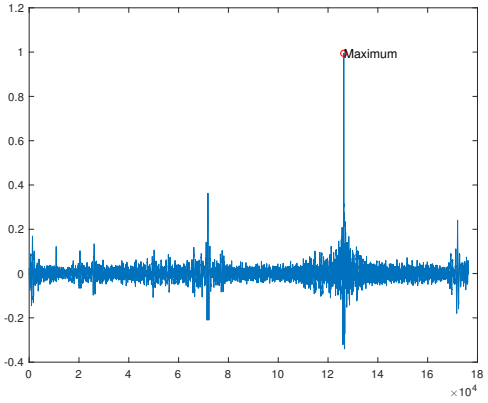


Figure 2.1: Cross correlation of two signals.

The x coordinate of maximum correlation is 50081, the length of signal is 176401 and sample rate is 44100. Applying the function above can 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 Normalized Spatial Cross Correlation in 2d

Till now, the cross correlation we used can only computer one dimensional signal. But in later work, for comparing images and get the correlation of them smoothly, it is necessary to extend the cross correlation to two dimensions. Similar with signals can be represented 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 "lag" t over A, compute for each "lag" the cross-correlation. Normalized cross correlation of two matrices defines as:

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

Where $\bar{t}$ is the mean of t, $\overline{A_{lag_x,lag_y}}$ 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

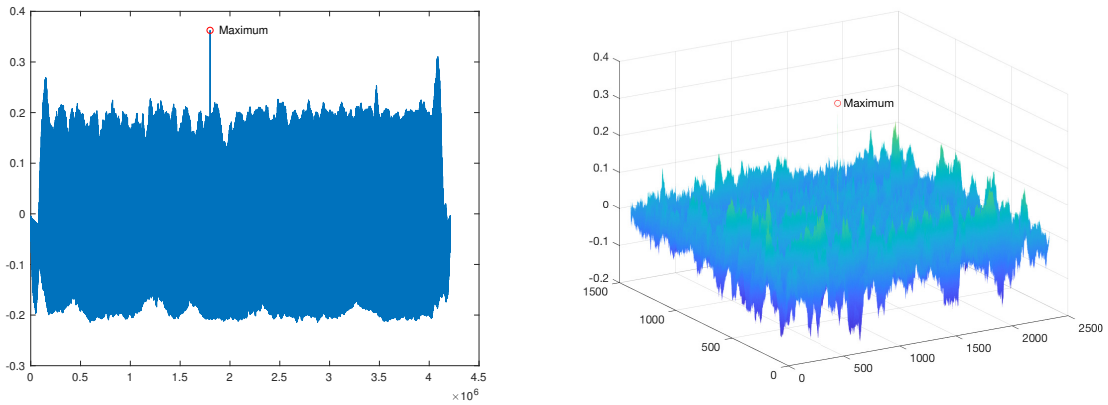
Images are just a matrix of pixel values in most image processing. It is naturally think of computing cross correlation between two images to get more relation informations of them. In this task, there are two images, one of them is a section of the other, so it is easy to find where the section of the 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.

(a) Rocket Man (section). **(b)** Maze (search region).

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 blue circle in Figure 2.2b.



(a) 2D visualization **(b)** 3D visualization

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.

The fast Fourier transform (FFT) algorithm can 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 $O(n \log n)$ [5].

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

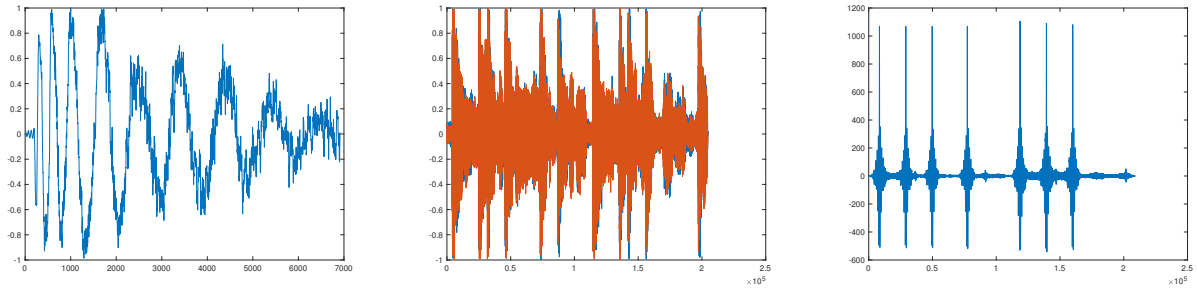
Source code of spectral cross correlation is in Appendix A.1.6.

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 2.4a and Figure 2.4b shows two pieces of waveforms separately. Figure 2.4c presents all occurrence of the element, as it is not a normalized correlation picture, all positions with the y-coordinates larger than 1000 is the appearance of the drum. In addition, other background noise have an effect on our calculation but not serious.

From the above plots, the signal length of one single drum is 7000, then cutting a 4.5 seconds music which length in Matlab is about $2 * 10^5$. There are 7 occurrences of the dum in this episode.

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



(a) drum waveform **(b)** music waveform **(c)** cross correlation

Figure 2.4: Two oscillographs

2.7 Cross correlation in three dimensions

This part is just some thinking by myself.

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 computed, 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 there are one cell with a particular disease as specimen and no one know the reason of its pathogenicity. In the meanwhile, for several cells, some of them 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 tool like today's 2d cross correlation in the future.

Well, there are already some research about 3D cross correlation[11][3].

Chapter 3

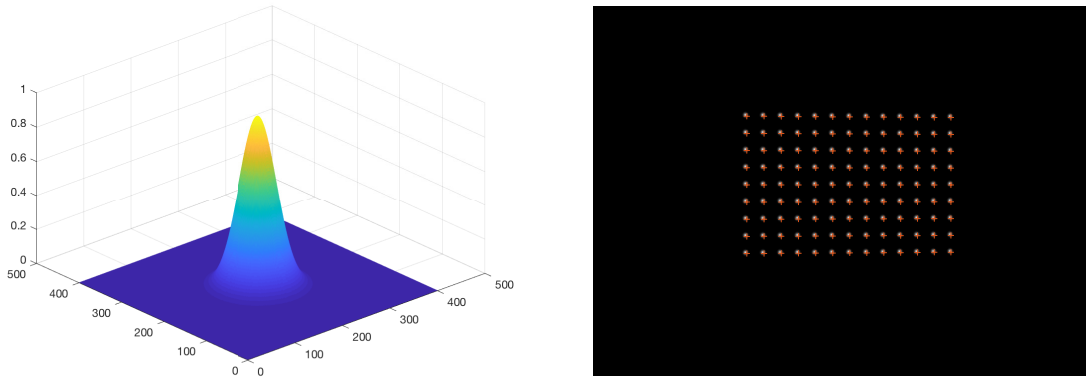
Calibration Model

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 optimizations are proposed together 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 method 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 computing cross correlation of Gaussian template and calibration plate, all dots can be found by local maximum method, that is, a peak found in one part of the result matrix each time and set the value in this area to 0, until no more peak can be detected. 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.



(a) 2D Gaussian **(b)** Dot detection results.

Figure 3.1: Dot detection. 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 red line reveals the order of the dots detected. However, we need a list of dots sorted from top

to bottom, left to right. Sorted dots shows 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 $\pm 9°$. The calibration target has white dots spaced by 50 mm in the x (horizontal) and y (vertical) directions and it is shifted to various z locations with the given distance that away from the cameras. Therefore the location of the dots in real place will be known as well as the location in the left and right images can also be detected in task 1. Figure 3.3 demonstrates the mechanism of calibration model. In this figure, O and O' are two cameras and X is the object.

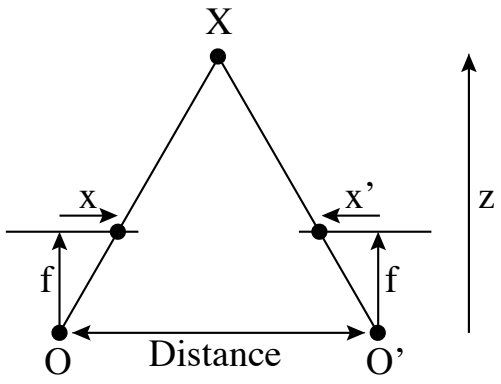


Figure 3.3: Calibration model mechanism and extrinsic parameters calculation.

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 cameras and object z and the distance between image planes and a lens (which provides a transformation between object space and image space) f are known, x and x' are available from two taken pictures. So we can calculate the distance between two cameras through triangle similarity theorems.

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

3.3 Image Comparison

After build calibration model. We hope to test the performance of our model using a pair of general images. But the images taken by cameras are not dots with certain distribution rules. Preprocess of the image is needed before applying it to our calibration model.

As observed in Chapter 2, the cross correlation technique 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 our computers can not thinking like human so that image comparison must be done for matching objects in the two views.

(a) Left view **(b)** Right view

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).

Firstly, let one image broken up in to windows and create a template from one window, like shown in Figure 3.4a, then creating a search region (larger than the template) in the other image as shown in Figure 3.4b. Additionally, scanning the template around the search region is able to find similar feature using cross correlation, in general as well as get the position of max correlation and compute the difference in pixel location (disparity). Repeat this for all windows and then we can get all corresponding positions which is very similar with the corresponding dots we detected in task 1.

The search region is 9 times larger than template in this task. However, for a more general task, exhaustively search the space of right view images ensure the best match to be found. But this method is time consuming. therefore, speeding up at the risk of possible failure to find the best match is used. Actually, there is better image comparison technique that makes use of spatial intensity gradient of the image[12], and more methods [13], [14].

The result of image comparison 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.

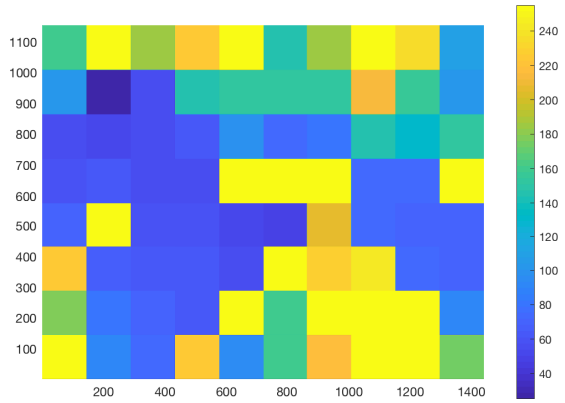


Figure 3.5: Image compare result, the region with color higher on the colorbar means the higher disparity

For better demonstrate image comparison. We compare two particular images collected from a standard stereo vision image set. Figure 3.6 shows the results. From which we can find the smaller windows size, the more information we can get. Another obvious phenomenon is the regions near to corner of the images tend to have high disparities cause these regions are pure black in left compared image so that it is hard to find a precise matched region in right image. Moreover, as the search region is set only 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.

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, 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 comparison, the left image will be split to several parts and each part will be searched in the corresponding but the larger part of the right image. Variable overlap is demonstrate in Figure 3.7, the split parts could have overlap. The most important benefit is the increase in 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 compared 4 times if there are 50% overlap, like the centre square of the right bottom part in Figure 3.7

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 out of it and we can not get any useful different information. However, if search region is too big, for example, treating the complete right view image as search region, it is fairly time consuming. So a proper size of search region is essential. On the other hand, a good shape of search region is also helpful, if our images are taken in the same horizontal level, we can

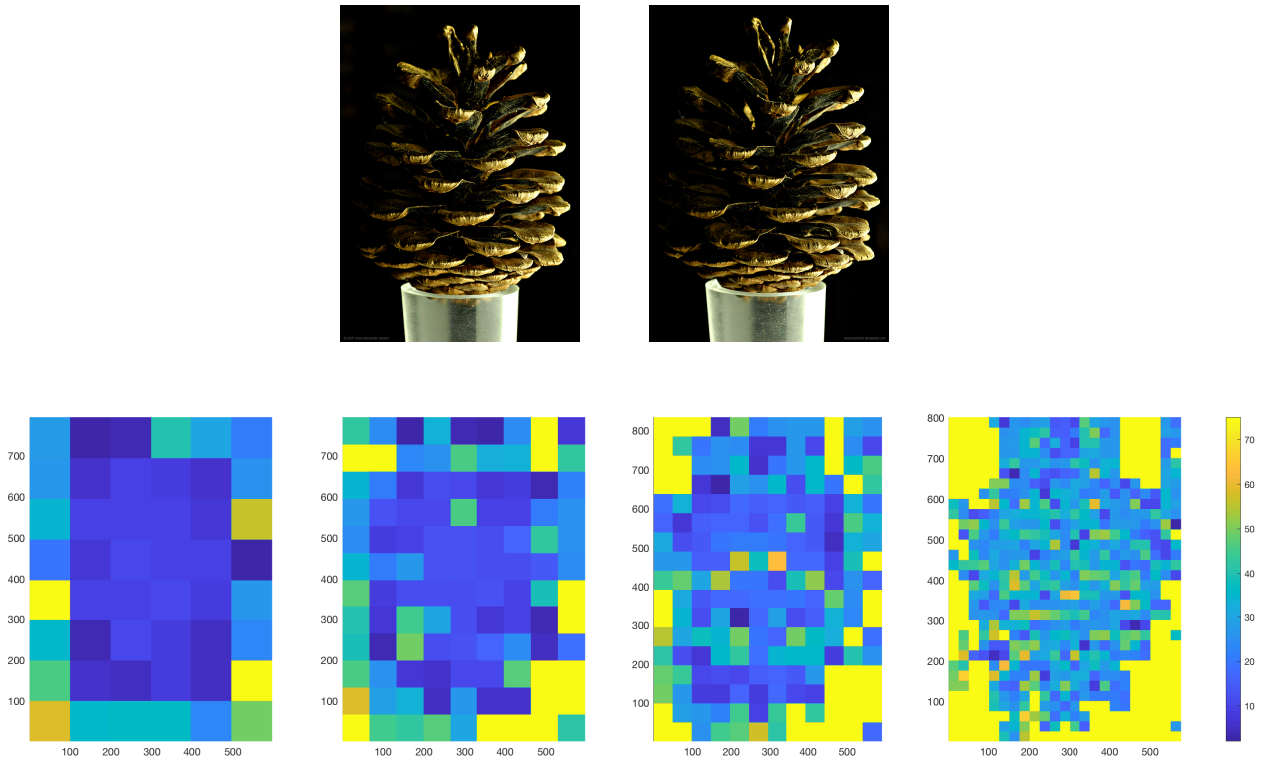


Figure 3.6: First two images are the image be compared, other four are results of different window size.

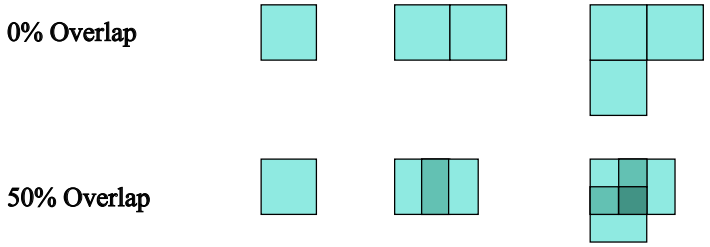


Figure 3.7: Example of variable window overlap

use flat search region rather than square region. Of course, the search region should extend in horizontal rather than vertical like shown in Figure 3.8, smaller search region could save time for us.

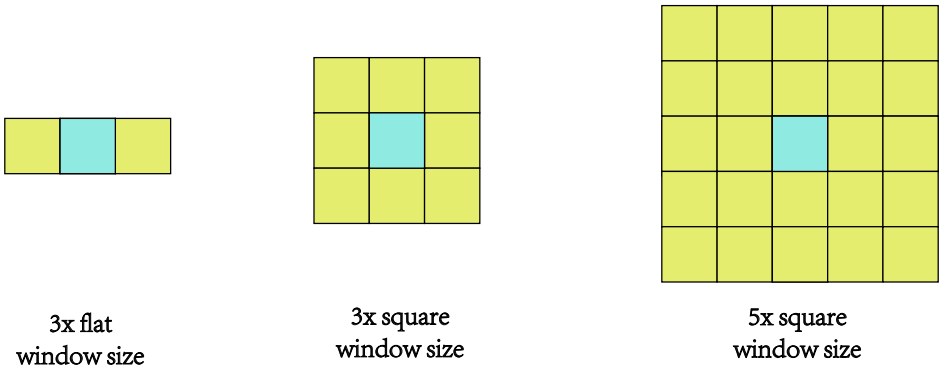
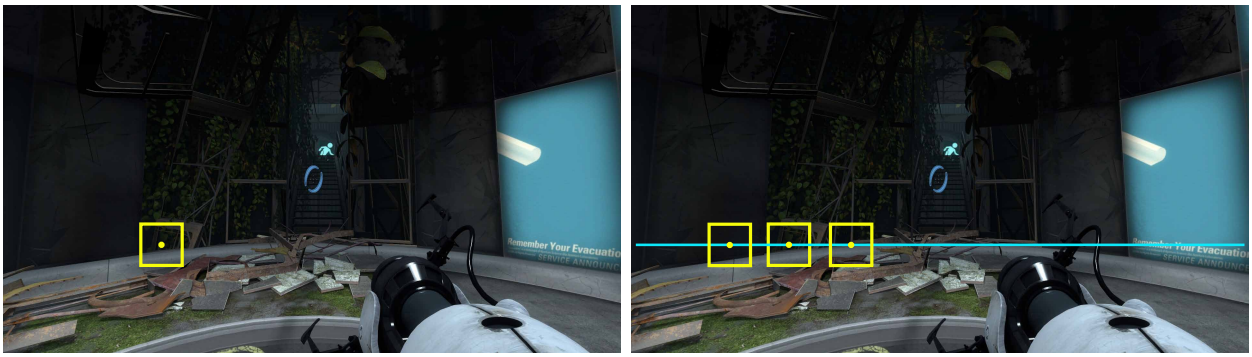


Figure 3.8: Example of variable search region

Actually, what image comparison do 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.



(a) Left view **(b)** Right view

Figure 3.9: Scanlines and more reasonable search region

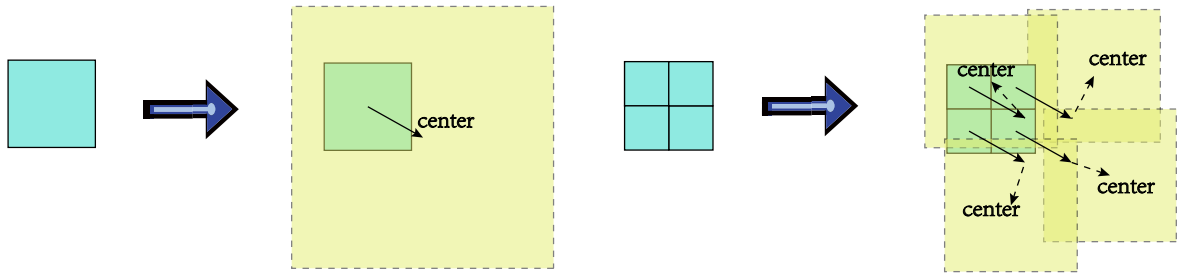
Source code of variable window overlap and variable search region geometry are in Appendix A.2.4.

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 centered 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 to 4 smaller templates and search region in the right image is centered at the location of the template plus (dpx, dpy) and it returns 4 new centers for next pass.



(a) 1st pass, wsize = 64 **(b)** 2nd pass, wsize = 32

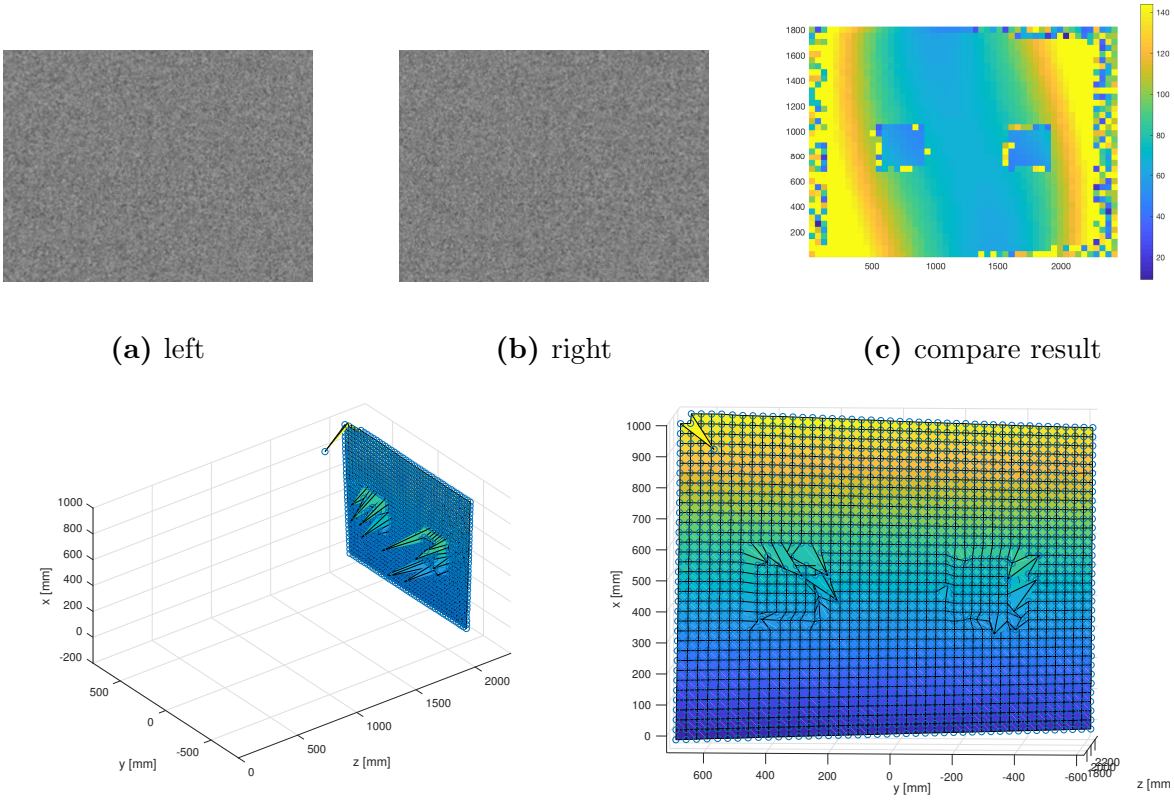
Figure 3.10: Multi-pass cross correlation. ("center" in the image refer to the new search region's center, yellow square represents new search regions.)

However, Multi-pass has its flaw. Once the first pass computes a bad shift distance, the second pass will be influenced by this and tend to get 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 one test image pairs. Figure 3.11a and Figure 3.11b are computer generated calibrated images from two views. Figure 3.11c is the image comparison result, Figure 3.11d is reconstructed objects in 3D space and Figure 3.11e is a clearer visualization of it. More results on other test sets are in Appendix B.



(a) left **(b)** right **(c)** compare result

(d) 3D reconstruction **(e)** 3D reconstruction

Figure 3.11: Test scan on computer generated calibrated images

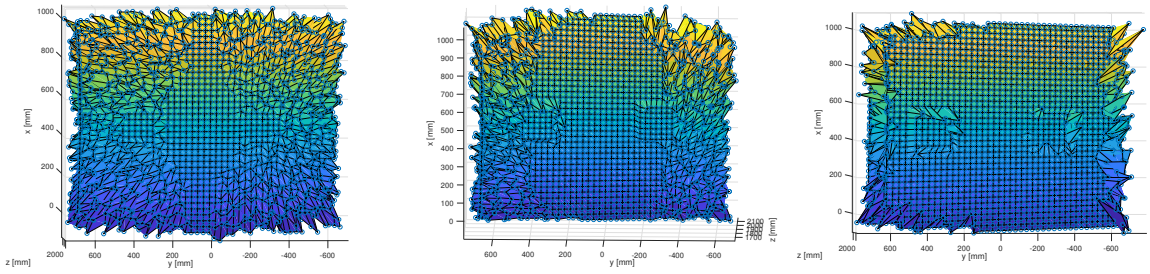
From the above images we find that the image comparison does not work well at the edge of images of the second and third test pairs. So we remove these vectors to get a more reasonable 3D reconstruction although there are still spurious exist.

Source code for 3D reconstruction 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 detection 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, some optimization optionals of reconstruction 3D scenery are investigated, Actually, the plots shown in Figure 3.11 are optimized reconstruction results. Figure 3.12 shows 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 if search region is small. 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 because the larger search region could return a more precise results, of course it costs more time.

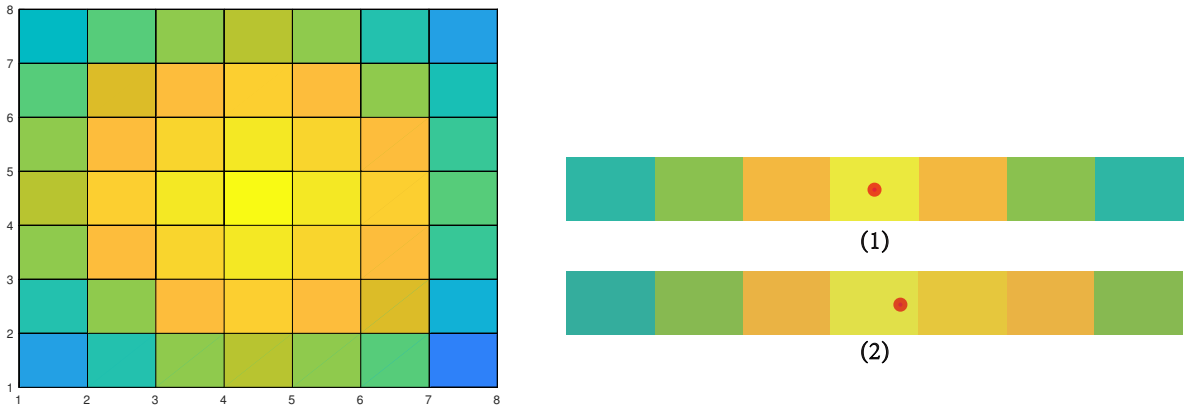


(a) 1.5x flat search region **(b)** 1.5x square search region **(c)** 3x square search region

Figure 3.12: Optimization of scan test

In addition to image comparison optimization, calibration model could be optimized through a more accurate dot detection algorithm, that is, sub-pixel level dot detection. Sub-pixel techniques have been was presented by Hueckel M F[7] and widely used in many image processing [1][4][8].

In this task, cross correlation X between calibration plate and gaussian template will be computed first. Different with dot detection in task 1, one peak detected in X (position with maximum value) would be treat as a part of one locking peak rather a detected dot. The whole part of locking peak is a range which center is the detected peak, as shown in Figure 3.13a. The central yellow square is the detected peak whose coordinate is $(4, 4)$ in this locked range (whole part). We compute the center of gravity weighted by the value of each pixel. Figure 3.13b demonstrate the strategy in one dimension. In Figure 3.13b(1), the center of gravity is coincide with the geometric center because the vector value is symmetrical. While in Figure 3.13b(2), the center of weight located on the right of the geometric center because the right part have higher value than left part, which gives them more weight. The method can easily extend to two dimensions and be used to find sub-pixel position of the maximum value.



(a) Surf plot of locked peak.

(b) Demonstration of center of gravity weighted by value, red dots is the center.

Figure 3.13: Sub-pixel dot detection method

Source code of sub-pixel level dot detection is in Appendix A.2.8.

3.7 Better Calibration model

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

The built calibration model is based 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 using several cameras. They do not have to be put in the same horizontal level, on the contrary, they can be put in the four corners 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 building calibration model, image comparison, to get the depth information. But do we really do this kind of calculations in our brain all the time? Do we estimate depth information simply 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 far larger than a person and when a building and a person project to our retina with the same size, we should know the build is far and the person is near to us.

Actually, our brain do 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.

4.1 Convolutional Neural Network

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

In this section. We build a model to classify the images of hand write number, 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.

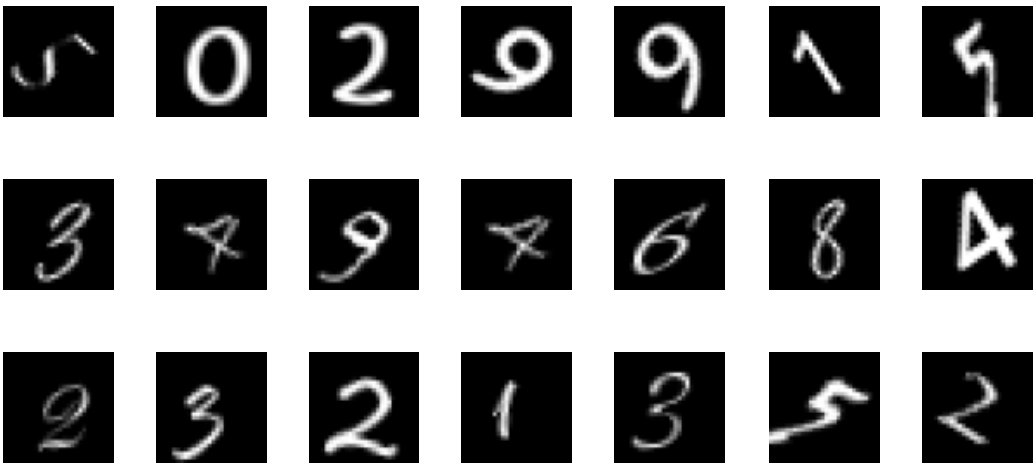


Figure 4.1: Hand write numbers

- 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.
- 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.

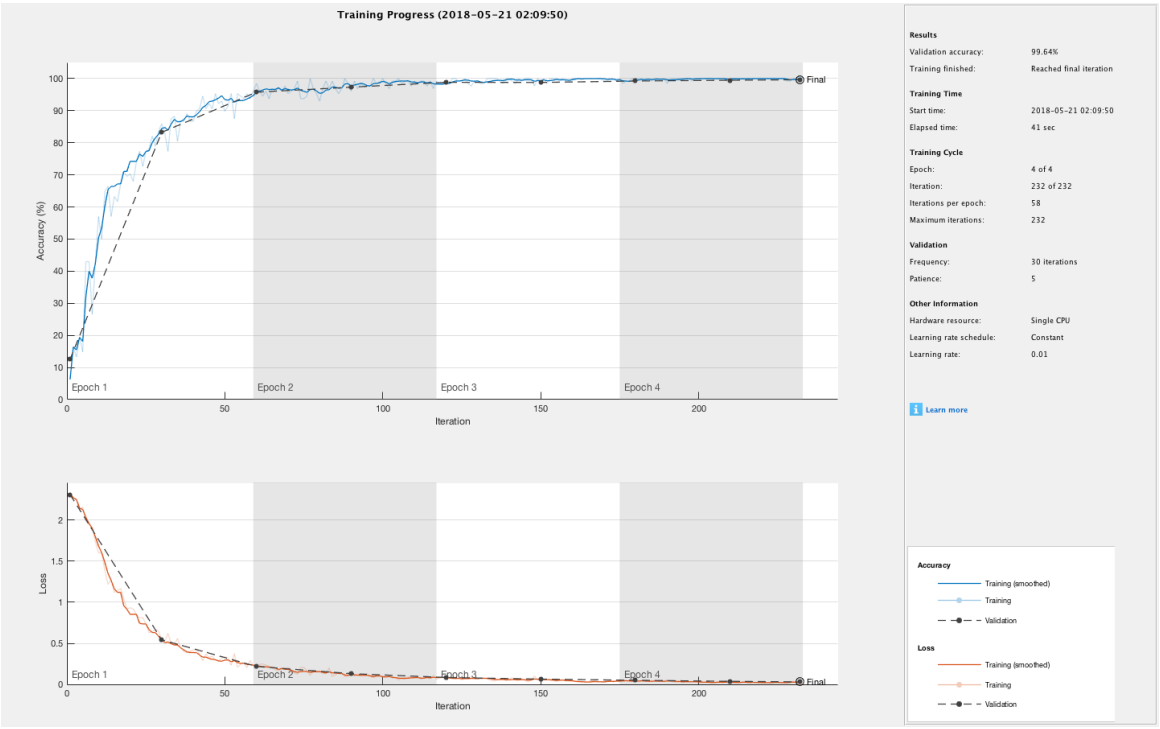


Figure 4.2: Training process

Source code for hand write image classifier is in Appendix A.3.1.

4.2 Stereo Convolutional Neural Network

This section 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 maxpooling layers followed by a series of upscalling 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 *.

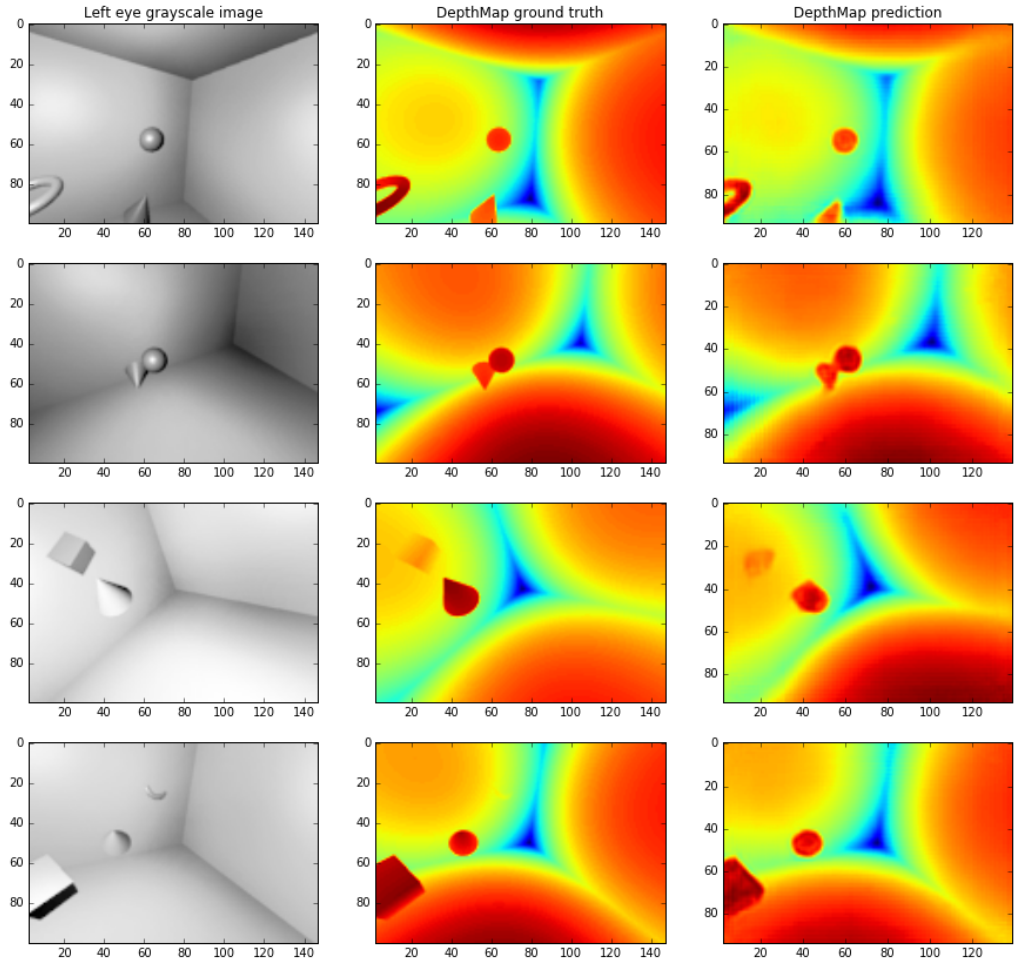


Figure 4.3: Stereo ConvNet results

There are many related works about Stereo ConvNet, basic architecture contains 4 convolution layers, 3 maxpool layers and 4 deconvolution layers†. To achieve better performance Deeper Stereo ConvNet Architecture [2] and Patched Deeper Stereo ConvNet Architecture [10][9] are proposed.

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

Appendix A

Matlab code

A.1 Part 1

A.1.1 Spatial Cross Correlation in 1d

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author:   Haonan Li                                                        %
% Purpose:  Compute the  spatial cross correlation of two vectors   %
%           with the same size                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function vec_res = spatial_correlation_1d(vec_a,vec_b)
%% padding 0 in the head and tail of vec_a
len = length(vec_a);
vec_a = vec_a - mean(vec_a);
vec_b = [zeros(1, len-1), vec_b - mean(vec_b), zeros(1, len-1)];

%% cross correlation
vec_res = zeros(1, 2*len-1);
for i = 1:(2*len-1)
    vec_res(i) = sum(vec_a .* (vec_b(1,i:i+len-1)))/len;
end
```

A.1.2 Normalized Spatial Cross Correlation in 1d

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author:   Haonan Li                                                        %
% Purpose:  Compute the normalised cross correlation of two vectors %
%           of the same size                                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function vec_res = normalised_spatial_correlation_1d(vec_a, vec_b)
%% normalization
len = length(vec_a);
vec_a = vec_a - mean(vec_a);
vec_b = vec_b - mean(vec_b);
% compute sigma
sigma = sqrt(sum(vec_a.^2)*sum(vec_b.^2))
% padding 0 in the head and tail of vec_b
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_corr = spatial_correlation_1d(sig_1, sig_2)';
20 % spectral method (faster)
21 cross_corr = 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_corr))
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
31
32 %% show
33 figure
34 plot(cross_corr)
35 ylim([-0.4,1.2])
36 hold on
37 plot(max_pos,max_value,'ro')
38 text(max_pos,max_value,'Maximum','FontSize',12)
```

A.1.4 Normalized Spatial Cross Correlation in 2d

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Author:   Haonan Li                                                      %
3  % Purpose:  Receive two matrix, t(template) and A(search region)    %
```

```
%           return normalized cross-correlation of them              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function mat_res = normalised_spatial_correlation_2d(mat_t, mat_A)
%% init mat_res with 0
[tY,tX] = size(mat_t);
[AY,AX] = size(mat_A);
mat_res = zeros(AY-tY+1, AX-tX+1);
[rY,rX] = size(mat_res);

%% cross correlation
% mat_t_ is a matrix of mat_t minus mean(mat_t)
mat_t_ = mat_t - mean(mat_t);
for i = 1:rY
    for j = 1:rX
        % compute mat_res(i,j)
        % first get the part of mat_A covered by the mat_t
        mat_A_under = mat_A(i:(i+tY-1), j:(j+tX-1));
        mat_A_under_ = mat_A_under - mean(mat_A_under);
        % compute numerator and denominator of the cross correlation
        numerator = sum(sum(mat_t_ .* mat_A_under_));
        denominator = sqrt(sum(sum(mat_t_.^2)) * sum(sum(mat_A_under_.^2)));
        mat_res(i,j) = numerator/denominator;
    end
end

%% figure
figure,
surf(mat_res);
figure(gcf)
```

A.1.5 Image Alignment

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author:   Haonan Li                                                    %
% Purpose:  Give two pictures, one is a part of the other, find the      %
%           corresponding position and mark it                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [pos_y, pos_x] = find_the_rocket_man(pic_t, pic_A)
%% read images
pic_t = imread(pic_t);
pic_A = imread(pic_A);
mat_t = mean(pic_t, 3);
mat_A = mean(pic_A, 3);
[tY,tX] = size(mat_t);

%% compute cross correlation and run time
tic;
% cross_corr = normalised_spatial_correlation_2d(mat_t,mat_A);
 cross_corr = my_norm_xcorr2(mat_A,mat_t);
run_time = toc

% max cross correlation position
[pos_y, pos_x] = find(cross_corr == max(max(cross_corr)));

%% plot cross correlation
```

```
figure
plot(cross_corr)
hold on

%% save marked image
imshow(pic_A);
hold on;
marker = plot(pic_A(1,1), 'o');
marker.XData = pos_x + tX/2;
marker.YData = pos_y + tY/2;
marker.MarkerSize = 50;
marker.LineWidth = 5;
marker.Color = 'b';
saveas(gcf, 're','png');
```

A.1.6 Spectral Cross Correlation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author:   Haonan Li                                                        %
% Purpose:  Compute the spectral cross correlation of two vectors   %
%           of the same size using Fourier transform.                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function vec_res = spectral_correlation_1d(vec_a, vec_b)

% transfer to spectral domain
f_a = fft(vec_a);
f_b = fft(vec_b);
% spectral cross correlation
conj_f_a = conj(f_a);
mid_res = conj_f_a .* f_b;
vec_res = ifft(mid_res);


%% My fft, does not work if N is not the power of 2
function res_fft = my_fft(vec)

vec_size = size(vec);
N = vec_size(2)
c = zeros(1,N);
% indexing computation
j1 = 0;
for i = 1 : N
    if i < j1 + 1
        tmp = vec(j1 + 1);
        vec(j1 + 1) = vec(i);
        vec(i) = tmp;
    end
    k = N / 2;
    while k ≤ j1
        j1 = j1 - k;
        k = k / 2;
    end
    j1 = j1 + k;
end

%% Butterfly computing
```

```
dig = 0;
k = N;
while k > 1
    dig = dig + 1;
    k = k / 2;
end

n = N / 2;
for m = 1 : dig
    dist = 2 ^ (m - 1);
    idx = 1;
    for i = 1 : n
        idx1 = idx;
        for j1 = 1 : N / (2 * n)
            r = (idx - 1) * 2 ^ (dig - m);
            coef = exp(j * (-2 * pi * r / N));
            tmp                 = vec(idx);
            vec(idx)          = tmp + vec(idx + dist) * coef;
            vec(idx + dist)   = tmp - vec(idx + dist) * coef;
            idx = idx + 1;
        end
        idx = idx1 + 2 * dist;
    end
    n = n / 2;
end
res_fft = vec;
```

A.1.7 Pattern Finder

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author:   Haonan Li                                                        %
% Purpose:  Find all accurance of a particular music element in a            %
%           piece of music                                                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function res = pattern_finder(music,pat)
%% load the music and a particular element (drum)
[y1,Fs1] = audioread(music);
x1 = y1(:,1);
[y2,Fs2] = audioread(pat);
x2 = y2(:,1);

%% cross correlation
corr_vec = xcorr(x1',x2');
res = corr_vec;

%% show
% plot(y1);
% plot(y2);
 plot(res);
```

A.2 Part 2

A.2.1 Dot Detection Algorithm

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author:   Haonan Li                                                          %
% Purpose:  Detect all dots on a calibration plate and sort them               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function dots = dot_detect(img_t)
%% read image
img = imread(img_t);
mat_img = rgb2gray(img);
[X,Y] = size(mat_img);

%% create a Gaussian template
u = [-2:2];
v = [-2:2];
[U,V] = meshgrid(u,v);
mat_gauss = exp(-(U.^2+V.^2)./2/3^2);
% draw the gaussian template
% [U,V] = meshgrid(u,v);
% mat_gauss = exp(-(U.^2+V.^2)./2/3^2);
% surf(mat_gauss), shading flat
% hold on

%% cross correlation
cross_corr = xcorr2(mat_img, mat_gauss);
% local max cross correlation position
dots = local_max(cross_corr,50);
% dots = order_dots(dots,21,17);
dots = order_dots(dots,13,9);

% draw the point in the picture
% imshow(img)
% hold on
% plot(dots(:,2),dots(:,1),'+','Markersize',10)
% % draw line to check the point order
% plot(dots(:,2),dots(:,1))
end


%% find all local maximum dotsm, level is the local range
function res = local_max(mat_a, level)
res = [];
[X,Y] = size(mat_a);
x = 1+level;
while x < X-level
    y = 1+level;
    while y < Y-level
        if mat_a(x,y) > 0
            local_maxx = max(max(mat_a(x-level:x+level,y-level:y+level)));
            if mat_a(x,y) == local_maxx
                res = [res;x,y];
                % if find one local max, set positions around it to 0
                mat_a(x-level:x+level,y-level:y+level) = 0;
            end
        end
        y = y + 1;
    end
    x = x + 1;
end
end

```

```
%% order the dots
function res = order_dots(dots, n, m)
res = dots;
for i=1:m
    for j=1:n
        st = (i-1)*n+1;
        en = i*n;
        res(st:en,:) = sortrows(dots(st:en,:),2);
    end
end
end
```

A.2.2 Create Calibration Model

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author:   Haonan Li                                                      %
% Purpose:  Create a calibration model use a set of calibration images     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [fitfunx, fitfuny, fitfunz] = calibration_model()
%% input all calibration images
real_dot = [];
left_dot = [];
right_dot = [];
for i = 0:5
    dis_z = 1900 + i * 20;
    left_img = ['Resources/cal_image_left_', num2str(dis_z), '.tiff'];
    right_img = ['Resources/cal_image_right_', num2str(dis_z), '.tiff'];
    real_dot = [real_dot; build_dot(dis_z)];
    left_dot = [left_dot; dot_detect(left_img)];
    right_dot = [right_dot; dot_detect(right_img)];
end

%% build fit functions
ly = left_dot(:,1);
lx = left_dot(:,2);
ry = right_dot(:,1);
rx = right_dot(:,2);
realy = real_dot(:,1);
realx = real_dot(:,2);
realz = real_dot(:,3);
capture = [lx,ly,rx,ry];

fitfunx = myfun(capture, realx);
fitfuny = myfun(capture, realy);
fitfunz = myfun(capture, realz);
end

%% compute dots positions in real space
function real = build_dot(real_z)
real = [];
for j = 1:17
    for i = 1:21
        index = 21*(j-1)+i;
        real(index,:) = [j*50,-500+i*50,real_z];
    end
```

```
end
end

%% output the calibration model
function func = myfun(capture, real)
fitfun = polyfitn(capture, real, 3);
sx     = (polyn2sym(fitfun));
func   = matlabFunction(sx);
end
```

A.2.3 Image Comparison

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author:   Haonan Li                                                      %
% Purpose:  Compare two images, split one image to several windows,        %
%           create template for each window and the search region          %
%           (larger than the template) in the other image, scan the        %
%           template around the search region and find the most            %
%           part, return the difference in pixel location                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function res = image_compare(img_a, img_b)
%% input images
img_a = imread(img_a);
A = rgb2gray(img_a);
img_b = imread(img_b);
B = rgb2gray(img_b);
[Y,X] = size(B);

%% image comparasion
% store the result
diff = A.*0;
res = [];
% split the larger side length to 10
wsize = round( max(X, Y) / 10);
for i = 0 : ceil(X/wsize) - 1
    for j = 0 : ceil(Y/wsize) - 1
        xgrid = 1 + i * wsize;
        ygrid = 1 + j * wsize;
        % find correspond points of two images
        [dx,dy] = corr_search(A, B, wsize, xgrid, ygrid);
        res = [res;xgrid,ygrid,xgrid+dx,ygrid+dy];
        diff(ygrid:(ygrid+wsize),xgrid:(xgrid+wsize)) = sqrt(dx*dx+dy*dy);
    end
end

%% show
ssize = 10*wsize;
diff(diff>ssize) = ssize;
surf(diff)
axis image
shading interp
colorbar
hold on
end


```

```
%% compute the difference in pixel location between template in left iamge
%  and most similar region in right image
function [dx,dy] = corr_search(mat_a, mat_b, wsize, xgrid, ygrid)
%% pattern (window)
[Y,X] = size(mat_a);
pat_left = xgrid;
pat_right = min(X,xgrid + wsize - 1);
pat_top = ygrid;
pat_bottom = min(Y,ygrid + wsize - 1);
pattern = mat_a(pat_top:pat_bottom, pat_left:pat_right);

%% search region: 3x window size square
sr_left = max(1, pat_left - wsize + 1);
sr_right = min(X, pat_right + wsize);
sr_top = max(1, pat_top - wsize + 1);
sr_bottom = min(Y, pat_bottom + wsize);
search_region = mat_b(sr_top:sr_bottom, sr_left:sr_right);

%% find the max cross correlation position
cross_corr = my_norm_xcorr2(search_region, pattern);
% cross_corr = normxcorr2(pattern, search_region);
[rel_y,rel_x] = find(cross_corr == max(cross_corr(:)));
if isempty(rel_x)
    dx = 1000;
    dy = 1000;
else
    dx = rel_x(1) + sr_left - pat_left-1;
    dy = rel_y(1) + sr_top - pat_top-1;
end
end

%% 2d cross correlation
function mat_r = my_norm_xcorr2(mat_A, mat_t)
[Ay, Ax] = size(mat_A);
[ty,tx] = size(mat_t);
% Change - Compute the cross power spectrum
Ga = fft2(mat_A);
Gb = fft2(mat_t, Ay, Ax);
mat_r = real(ifft2((Ga.*conj(Gb))./abs(Ga.*conj(Gb))));
end
```

A.2.4 Cross Correlation Optimization

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author:   Haonan Li                                                      %
% Purpose:  Optimization of image compare, overlap refers window    %
%           overlap. Range in [0,1), sr_size is the what times       %
%           search size of window, search shape values [s]: square  %
%           or [f]: flat.                                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function res = image_compare_optimized(img_a, img_b, overlap, sr_size, ...
    sr_shape)
%% input images
img_a = imread(img_a);
A = rgb2gray(img_a);
img_b = imread(img_b);
```

```
B = rgb2gray(img_b);
[Y,X] = size(B);

%% image comparation
res = [];
diff = A.*0;
% decide window size
wsize = round( max(X, Y) / 50 );
% compute shift size by overlap
shift = ceil( wsize * (1 - overlap));
ceil((X-wsize)/shift)
ceil((Y-wsize)/shift)
for i = 0 : ceil((X-wsize)/shift)
    for j = 0 : ceil((Y-wsize)/shift)
        xgrid = 1 + i * shift;
        ygrid = 1 + j * shift;
        % find correspond points of two images
        [dx,dy] = corr_search(A, B, wsize, xgrid, ygrid, sr_size, sr_shape);
        res = [res;xgrid,ygrid,xgrid+dx,ygrid+dy];
        diff(ygrid:(ygrid+wsize),xgrid:(xgrid+wsize)) = sqrt(dx*dx+dy*dy);
    end
end

%% draw the corresponding points
diff(diff>3*wsize) = 3*wsize;
surf(diff)
axis image
shading interp
colorbar
hold on
end


%% optimized compare, two options (ovarlap and search region shape)
function [dx,dy] = corr_search(A, B, wsize, xgrid, ygrid, sr_size, sr_shape)
%% pattern (window)
[Y,X] = size(B);
pat_left = xgrid;
pat_right = min(X,xgrid + wsize - 1);
pat_top = ygrid;
pat_bottom = min(Y,ygrid + wsize - 1);
pattern = A(pat_top:pat_bottom, pat_left:pat_right);

%% search region
extend = floor(wsize * (sr_size - 1)/2);
sr_left = max(1, pat_left - extend + 1);
sr_right = min(X, pat_right + extend);
% search region shape
if sr_shape == 's' % square
    sr_top = max(1, pat_top - extend + 1);
    sr_bottom = min(Y, pat_bottom + extend);
elseif sr_shape == 'f' % flat
    sr_top = pat_top;
    sr_bottom = pat_bottom;
end
search_region = B(sr_top:sr_bottom, sr_left:sr_right);

%% find the max cross correlation position
cross_corr = normxcorr2(pattern, search_region);
[rel_y,rel_x] = find(cross_corr == max(cross_corr(:)));
```

```
if isempty(rel_x)
    dx = 0;
    dy = 0;
else
    dx = rel_x(1) + sr_left - pat_left-1;
    dy = rel_y(1) + sr_top - pat_top-1;
end
end

%% 2d cross correlation
function mat_r = my_norm_xcorr2(mat_A, mat_t)
[Ay, Ax] = size(mat_A);
[ty,tx] = size(mat_t);
% Change - Compute the cross power spectrum
Ga = fft2(mat_A);
Gb = fft2(mat_t, Ay, Ax);
mat_r = real(ifft2((Ga.*conj(Gb))./abs(Ga.*conj(Gb))));
end
```

A.2.5 Multi-pass Cross Correlation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author:   Haonan Li                                                        %
% Purpose:  Multipass image comparison: recursively call image              %
%           comparison function of part of the whole image                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function diff = image_compare_multipass(img_a, img_b)
%% input images
img_a = imread(img_a);
A = rgb2gray(img_a);
img_b = imread(img_b);
B = rgb2gray(img_b);
[Y,X] = size(B);

%% image comparison
% store the result
diff = A.*0;
% init window size
wsize = 128;
% recersive level
level = 2;
for i = 0 : ceil(X/wsize) - 1
    for j = 0 : ceil(Y/wsize) - 1
        xgrid = 1 + i * wsize;
        ygrid = 1 + j * wsize;
        % find correspond points of two images
        diff = img_cmp(A, B, wsize, xgrid, ygrid, 0, 0, diff, 2);
    end
end

%% show
ssize = 3*wsize/2^level;
diff(diff>ssize) = ssize;
surf(diff)
axis image
shading interp
```

```
colorbar
hold on
end

%% recursive compare function
function diff = img_cmp(A, B, wsize, xgrid, ygrid, dpx, dpy, diff, level)
%% pattern (window)
[Y,X] = size(A);
pat_left = xgrid;
pat_right = min(X,xgrid + wsize - 1);
pat_top = ygrid;
pat_bottom = min(Y,ygrid + wsize - 1);
pattern = A(pat_top:pat_bottom, pat_left:pat_right);

%% search region: 3x window size square
sr_left = max(1, pat_left - wsize + 1 + dpx);
sr_right = min(X, pat_right + wsize + dpx);
sr_top = max(1, pat_top - wsize + 1 + dpy);
sr_bottom = min(Y, pat_bottom + wsize + dpy);
% if search region out of bound completely, stop
if sr_left > X | sr_top > Y
    diff(pat_top:pat_bottom,pat_left:pat_right) = sqrt(2000000);
    return
end
search_region = B(sr_top:sr_bottom, sr_left:sr_right);

% find the max cross correlation position
cross_corr = my_norm_xcorr2(search_region, pattern);
[rel_y,rel_x] = find(cross_corr == max(cross_corr(:)));
% can not find a cross corr result in big picture
if isempty(rel_x) & level > 1
    dx = 0;
    dy = 0;
elseif isempty(rel_x) & level == 1
        dx = 1000;
        dy = 1000;
else
    dx = rel_x(1) + sr_left - pat_left-1;
    dy = rel_y(1) + sr_top - pat_top-1;
end
diff(pat_top:pat_bottom,pat_left:pat_right) = sqrt(dx*dx+dy*dy);
% if level == 1, stop recursive
if level == 1
    return
else
    wsize = wsize / 2;
end

%% recursively call image compare
level = level-1;
xgrid1 = pat_left;
xgrid2 = pat_left+wsize;
ygrid1 = pat_top;
ygrid2 = pat_top+wsize;
diff = img_cmp(A, B, wsize, xgrid1, ygrid1, dx, dy, diff, level);
if xgrid2 < X
    diff = img_cmp(A, B, wsize, xgrid2, ygrid1, dx, dy, diff, level);
end
if ygrid2 < Y
    diff = img_cmp(A, B, wsize, xgrid1, ygrid2, dx, dy, diff, level);
```

```
end
if xgrid2 < X & ygrid2 < Y
    diff = img_cmp(A, B, wsize, xgrid2, ygrid2, dx, dy, diff, level);
end
end

%% 2d cross correlation
function mat_r = my_norm_xcorr2(mat_A, mat_t)
[Ay, Ax] = size(mat_A);
[ty,tx] = size(mat_t);
% Change - Compute the cross power spectrum
Ga = fft2(mat_A);
Gb = fft2(mat_t, Ay, Ax);
mat_r = real(ifft2((Ga.*conj(Gb))./abs(Ga.*conj(Gb))));
end
```

A.2.6 Test Scan

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author:   Haonan Li                                                        %
% Purpose:  create a 3D reconstruction of the test image pairs               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function res = test_scan1(img_a, img_b, fx, fy, fz)
%% compare two images
fprintf("Image comparing...\n ");
left_dot = dot_detect(img_a);
right_dot = dot_detect(img_b);
cmp = [left_dot(:,2),left_dot(:,1),right_dot(:,2),right_dot(:,1)];
fprintf("Image comparing done!\n ");

%% init calibrate model (it may take a long time, so we make it parameters)
fprintf("Calibration model initializing...\n ");
% [fx, fy, fz] = calibration_model();
fprintf("Calibration model initialize done!\n ");

%% create a 3D reconstruction
[X,Y] = size(cmp)
res = [];
for i = 1:X
    realx = fx(cmp(i,1),cmp(i,2),cmp(i,3),cmp(i,4));
    realy = fy(cmp(i,1),cmp(i,2),cmp(i,3),cmp(i,4));
    realz = fz(cmp(i,1),cmp(i,2),cmp(i,3),cmp(i,4));
    res = [res;cmp(i,:),realx,realy,realz];
end

%% Plot
figure;
plot3(res(:,7),res(:,5),res(:,6),'o')
hold on
surf(reshape(res(:,7),[13,9]),reshape(res(:,5),[13,9]),reshape(res(:,6),[13,9]))
xlabel('z [mm]')
ylabel('y [mm]')
zlabel('x [mm]')
axis equal
axis([0,2000, -500,500, 0,800]);
grid on
```

A.2.7 Optimized Test Scan

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author:   Haonan Li                                                        %
% Purpose:  create a 3D reconstruction of the test image pairs               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function res = test_scan(img_a, img_b, fx, fy, fz)
%% compare two images
fprintf("Image comparing...\n ");
cmp = image_compare_optimized(img_a,img_b,0,5,'s');
size(cmp)
fprintf("Image comparing done!\n ");

%% init calibrate model (it may take a long time, so we make it parameters)
fprintf("Calibration model initializing...\n ");
% [fx, fy, fz] = calibration_model();
fprintf("Calibration model initialize done!\n ");

%% create a 3D reconstruction
[X,Y] = size(cmp)
res = [];
for i = 1:X
    realx = fx(cmp(i,1),cmp(i,2),cmp(i,3),cmp(i,4));
    realy = fy(cmp(i,1),cmp(i,2),cmp(i,3),cmp(i,4));
    realz = fz(cmp(i,1),cmp(i,2),cmp(i,3),cmp(i,4));
    res = [res;cmp(i,:),realx,realy,realz];
end

%% Plot
figure;
plot3(res(:,7),res(:,5),res(:,6),'o')
hold on
fsize = [38,51];
surf(reshape(res(:,7),fsize),reshape(res(:,5),fsize),reshape(res(:,6),fsize))
xlabel('z [mm]')
ylabel('y [mm]')
zlabel('x [mm]')
axis equal
% axis([0,2300, -800,800, -200,1000]);
grid on
```

A.2.8 Sub-pixel Dot Detection

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author:   Haonan Li                                                        %
% Purpose:  Detect dots on a calibration plate on a sub-pixel level         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function dots = subpixel_dot_detect(img_t)
%% read image
img = imread(img_t);
mat_img = rgb2gray(img);
[X,Y] = size(mat_img);

%% create a Gaussian template
```

```
u = [-1:1];
v = [-1:1];
[U,V] = meshgrid(u,v);
mat_gauss = exp(-(U.^2+V.^2)./2/3^2);

%% cross correlation
cross_corr = xcorr2(mat_img, mat_gauss);
% local max cross correlation position
dots = local_max(cross_corr,mat_gauss,3);
% dots = order_dots(dots,21,17);
size(dots)
dots = order_dots(dots,13,9);

% draw the point in the picture
% imshow(img)
% hold on
% plot(dots(:,2),dots(:,1),'+','Markersize',2)
end


%% find all local maximum dotsm, level is the local range
function res = local_max(cross_corr, mat_gauss, level)
    res = [];
    max_v = max(cross_corr(:));
    while max_v > 0
        [peak_y,peak_x] = find(cross_corr == max_v);
        y = peak_y(1);
        x = peak_x(1);
        lock_peak = cross_corr(y-level:y+level+1,x-level:x+level+1);
        % draw
        % surf(lock_peak)
        % hold on
        % pause(5)
        [dy,dx] = sub_pixel_peak(lock_peak);
        res = [res;y+dy,x+dx];
        cross_corr(y-50:y+50,x-50:x+50) = 0;
        max_v = max(cross_corr(:));
    end
end

%% find sub-pixel peak
function [dy,dx] = sub_pixel_peak(A)
    [Y,X] = size(A);
    S = sum(A(:));
    S_Y = 0;
    for i = 1:X
        S_Y = S_Y + i*sum(A(i,:));
    end
    dy = S_Y/S;
    S_X = 0;
    for i = 1:Y
        S_X = S_X + i*sum(A(:,i));
    end
    dx = S_X/S;
end

%% order the dots
function res = order_dots(dots, n, m)
    res = dots;
    for i=1:m
```

```
        for j=1:n
            st = (i-1)*n+1;
            en = i*n;
            res(st:en,:) = sortrows(dots(st:en,:),2);
        end
    end
end
```

A.3 Part 3

A.3.1 Convolutional Neural Network

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author:   Haonan Li, reference: MATLAB Documentation                        %
% Purpose:  Train a classifier for hand write number recogonize               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Load datasets
dataset = fullfile(matlabroot,'toolbox','nnet','nndemos', ...
    'nndatasets','DigitDataset');
imgs = imageDatastore(dataset, ...
    'IncludeSubfolders',true,'LabelSource','foldernames');

%% display some iamges
% figure;
% perm = randperm(10000,15);
% for i = 1:15
%     subplot(3,5,i);
%     imshow(imgs.Files{perm(i)});
% end

% get the size of images, it will use in input layer
img = readimage(imgs,1);
size(img);
% count label for spliting training and test set
labelCount = countEachLabel(imds);

%% training and test set
train_size = 750;
[img_train,img_test] = splitEachLabel(imgs,train_size,'randomize');

%% network architecture
layers = [
    imageInputLayer([28 28 1])

    convolution2dLayer(3,8,'Padding',1)
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,16,'Padding',1)
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)
```

```
45
46      convolution2dLayer(3,32,'Padding',1)
47      batchNormalizationLayer
48      reluLayer
49
50      fullyConnectedLayer(10)
51      softmaxLayer
52      classificationLayer];
53
54  % training options
55  options = trainingOptions('sgdm', ...
56      'MaxEpochs',4, ...
57      'ValidationData',img_test, ...
58      'ValidationFrequency',30, ...
59      'Verbose',false, ...
60      'Plots','training-progress');
61
62  %% train network
63  net = trainNetwork(img_train,layers,options);
64
65  % result
66  pred_label = classify(net,img_test);
67  test_label = img_test.Labels;
68
69  accuracy = sum(pred_label == test_label)/numel(test_label)
```

Appendix B

Test Scan Result

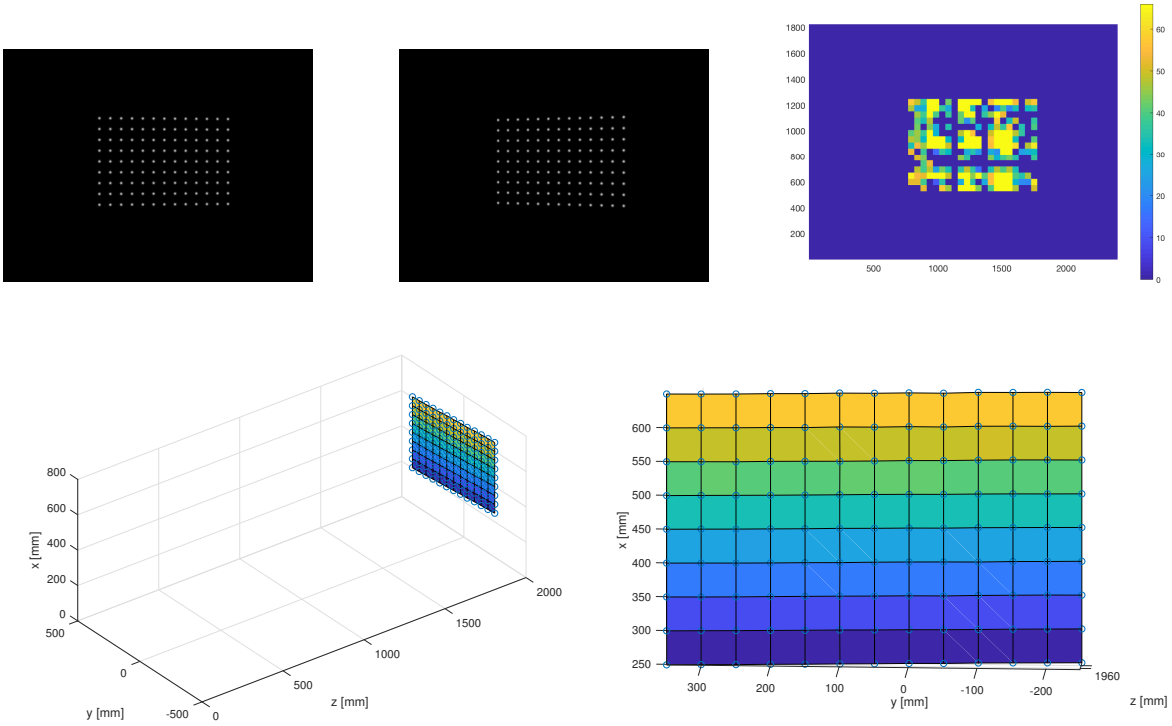


Figure B.1: Test scan on image pair 1

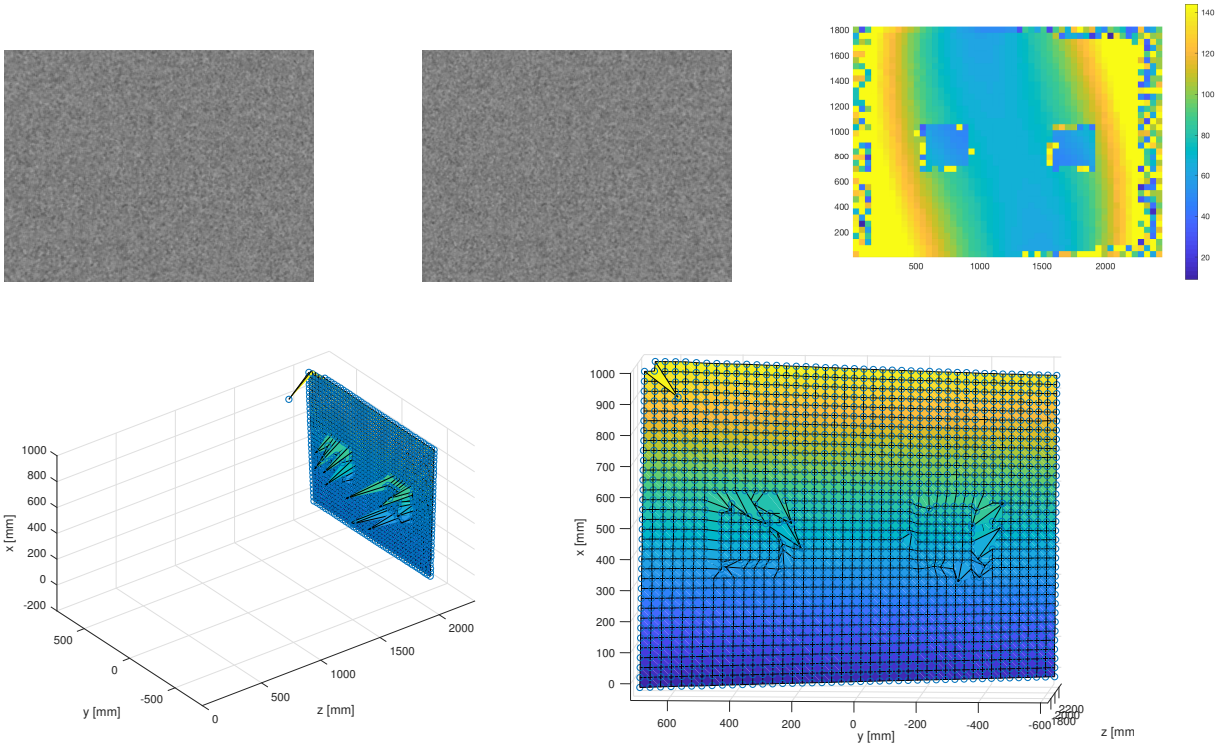


Figure B.2: Test scan on image pair 2

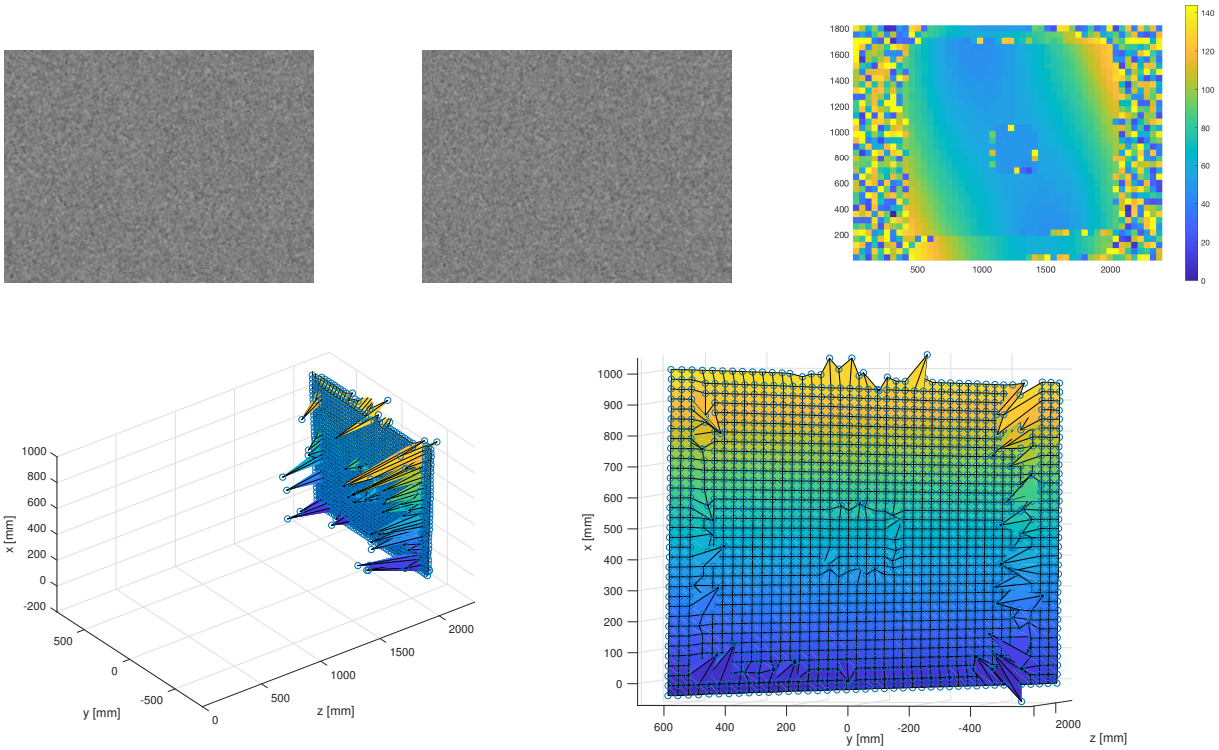


Figure B.3: Test scan on image pair 3

Bibliography

[1] G. Avrahami and V. Pratt. Sub-pixel edge detection in character digitization. In *Conference proceedings on Raster imaging and digital typography II*, pages 54–64, 1991.

[2] A. Dosovitskiy, P. Fischery, E. Ilg, and P. Usser. Flownet: Learning optical flow with convolutional networks. In *IEEE International Conference on Computer Vision*, pages 2758–2766, 2015.

[3] G. Haberfehlner. 3d nanoimaging of semiconductor devices and materials by electron tomography. 09 2013.

[4] H. T. Haskett. Ares method of sub-pixel target detection, 2001.

[5] M. Heideman, D. Johnson, and C. Burrus. Gauss and the history of the fast Fourier transform. *IEEE ASSP Magazine*, 1(4):14–21, October 1984.

[6] I. P. Howard and B. J. Rogers. *Binocular vision and stereopsis.* Oxford University Press, 1995.

[7] M. H. Hueckel. *A Local Visual Operator Which Recognizes Edges and Lines.* ACM, 1973.

[8] A. Huertas and G. Medioni. Detection of intensity changes with subpixel accuracy using laplacian-gaussian masks. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, PAMI-8(5):651–664, 2009.

[9] Y. Lecun. Stereo matching by training a convolutional neural network to compare image patches. 17(1):2287–2318, 2015.

[10] B. Li, C. Shen, Y. Dai, A. van den Hengel, and M. He. Depth and surface normal estimation from monocular images using regression on deep features and hierarchical crfs. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 00, pages 1119–1127, June 2015.

[11] B. L. O. H. S. D. N. G. K. E. K. Li Fan, JianZhong Qian. Automatic segmentation of pulmonary nodules by using dynamic 3d cross-correlation for interactive cad systems, 2002.

[12] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.

[13] D. Marr and T. Poggio. A computational theory of human stereo vision. *Readings in Cognitive Science*, 204(1156):301–328, 1988.

[14] H. P. Moravec. *Obstacle avoidance and navigation in the real world by a seeing robot rover.* Stanford University, 1980.

[15] L. Rabiner and B. Gold. *Theory and application of digital signal processing.* Prentice-Hall signal processing series. Prentice-Hall, 1975.

[16] L. Rabiner and R. Schafer. *Digital Processing of Speech Signals.* Prentice-Hall signal processing series. Prentice-Hall, 1978.