



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

Abstract

Contents

1	Introduction	3
2	Cross Correlations	4
2.1	Normalized Spatial Cross Correlation in 1d	4
2.2	Signal Offset	5
2.3	Normalised Spatial Cross Correlation in 2d	5
2.4	Image Alignment	5
2.5	Spectral Cross Correlation	6
2.6	Pattern Finder	8
2.7	Cross correlation in three dimensions	8
3		9
3.1	Dot Detection Algorithm	9
3.2	Create Calibration Model	9
3.3	Image Comparision	10
3.4	Cross Correlation Optimisation	10
Appendix A	Matlab code	13
A.1	Part 1	13
Bibliography		17

Chapter 1

Introduction

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 learn the cross correlation in 1 dimension and its applications to signal processing. Then we extend it to 2 dimensions and familiar its usage in image processing.

After these basic concepts and application, we consider the efficiency of computer cross correlation. We find it is really slow as the signals and images become larger, then we apply a new method as an alternative to spatial method, which is spectral cross correlation with a Fourier transform and inverse Fourier transform. We find this 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. As we all know, signal could be continuous and discrete. In this project, we only focus discrete signal.

For discrete functions f and g , the cross-correlation is defined as:

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

The problem of the above function is that the value of r is somewhat arbitrary because of the variant amplitude of f and g . 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

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

Matlab code of normalized 1d cross correlation is in Appendix A.1.1.

2.2 Signal Offset

If there are two signals (denote by vector) come from the same source and are just offset by some time. We can easily use cross correlation 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. 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.

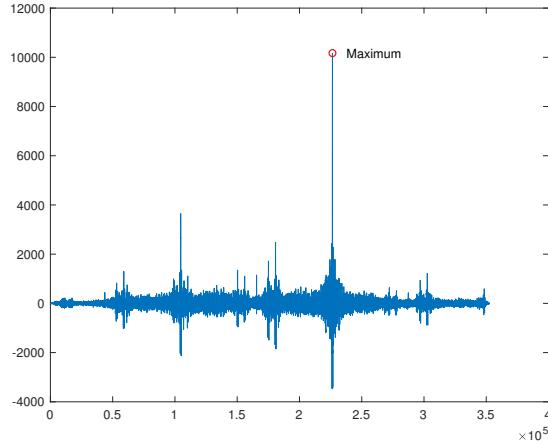


Figure 2.1: Cross correlation of two signals.

Matlab code of signal offset computing is in Appendix A.1.2.

2.3 Normalised Spatial Cross Correlation in 2d

The cross correlation can be extended to two-dimensional 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:

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

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

Matlab code of normalized 2d cross correlation is in Appendix A.1.3.

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.

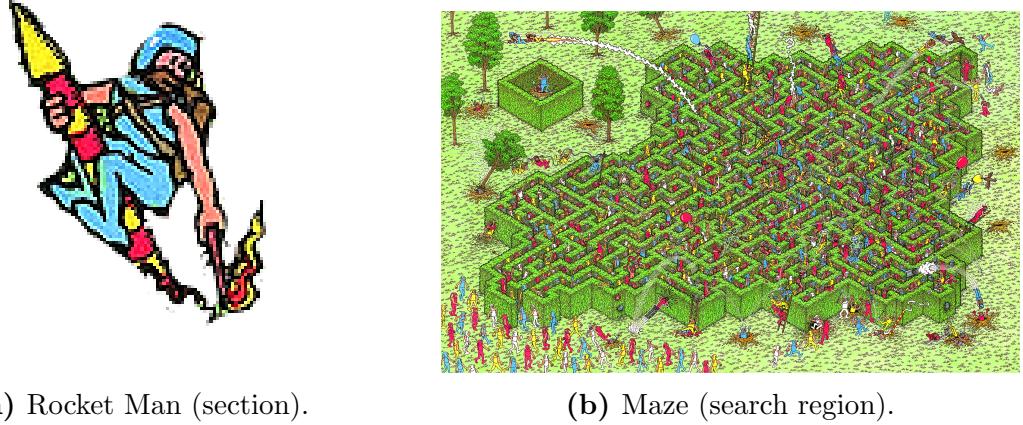


Figure 2.2: Image alignment.

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.

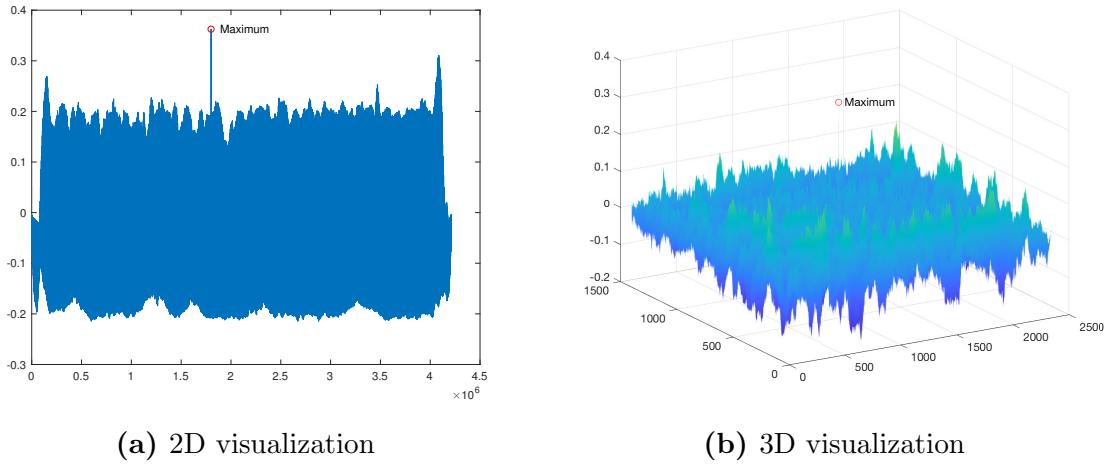


Figure 2.3: Cross correlation result matrix visualization

Matlab code of image algnment is in Appendix A.1.4.

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.

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 $O(n \log n)$.



Figure 2.4: Image Alignment result.

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

Matlab code of spectral cross correlation is in Appendix A.1.5.

2.6 Pattern Finder

In this task, I pick a piece of music and cut a small piece of it (a drum) through listening. Figure 2.5 shows two piece 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.

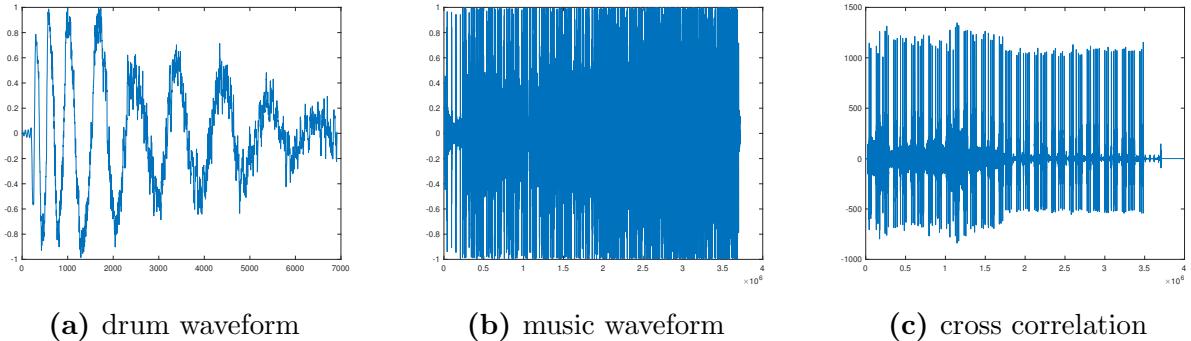


Figure 2.5: Two oscillographs

Matlab code of pattern finder is in Appendix A.1.6.

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 reearch 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

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. 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.

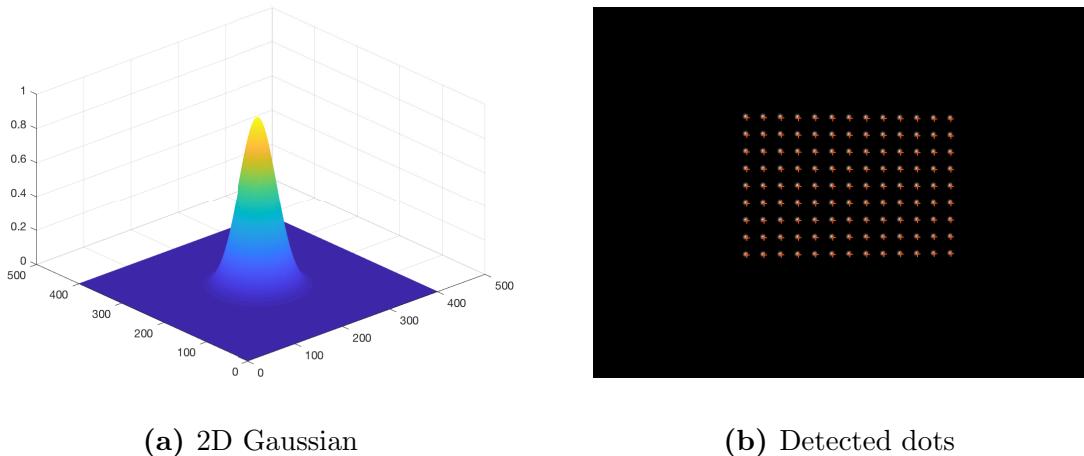


Figure 3.1: Dot Dectection.

Detect all dots is not the end of this task. For we need to use these dots in the later work. The points are now stored in the order of its been detected, shown in Figure 3.2a, the read line describe its storage order. We sort it in order for further using shown like Figure 3.2b.

3.2 Create Calibration Model

Our next step is to create a program which imports all the calibration images and records the pixel and real locations of all of the dots in each image. 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



(a) Disordered dots

(b) Ordered dots

Figure 3.2: Dot Storage.

and right images can also be detected in task 1. We finally find three functions to fit x , y and z coordinates in real space by using corresponding calibration target's coordinates in left and right images, as

$$\begin{aligned}x_{real} &= f_1(x_{left}, y_{left}, x_{right}, y_{right}) \\y_{real} &= f_2(x_{left}, y_{left}, x_{right}, y_{right}) \\z_{real} &= f_3(x_{left}, y_{left}, x_{right}, y_{right})\end{aligned}$$

3.3 Image Comparision

After we have built a calibration model. We want to apply it to practical scenes. But in practice, the images taken 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 taken by left and right cameras, find corresponding objects in two images.

As we 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 Melbourne University as shown in Figure 3.3. As humans, most of us can easily recognise that both images are of the same scene, only taken from different angles or positions. But we must do image comparison to find corresponding objects in two images and then apply it to calibration model.

We first break up one image into windows and create a template from one window, like shown in Figure 3.3a, then create a search region (larger than the template) in the other image as shown in Figure 3.3b. 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. 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.

3.4 Cross Correlation Optimisation

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 will investigate three optimisation strategies:

- Window overlap



(a) Left view

(b) Right view

Figure 3.3: Template created in left image (orange), search region created in right image (3 times larger than template, centered at the same pixel location).

- Search region

- Multiple pass

a). Variable window overlap

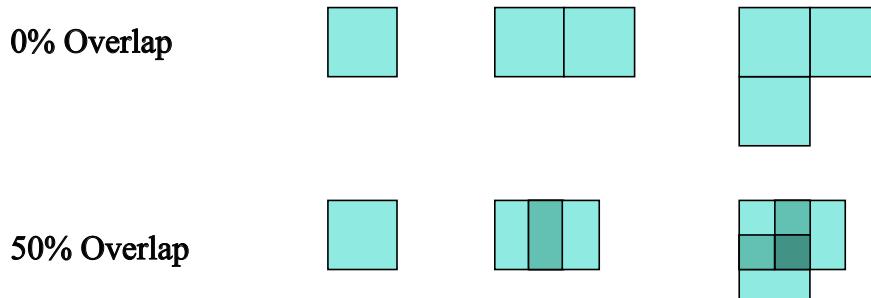


Figure 3.4: Example of variable window overlap

b). Variable search region geometry

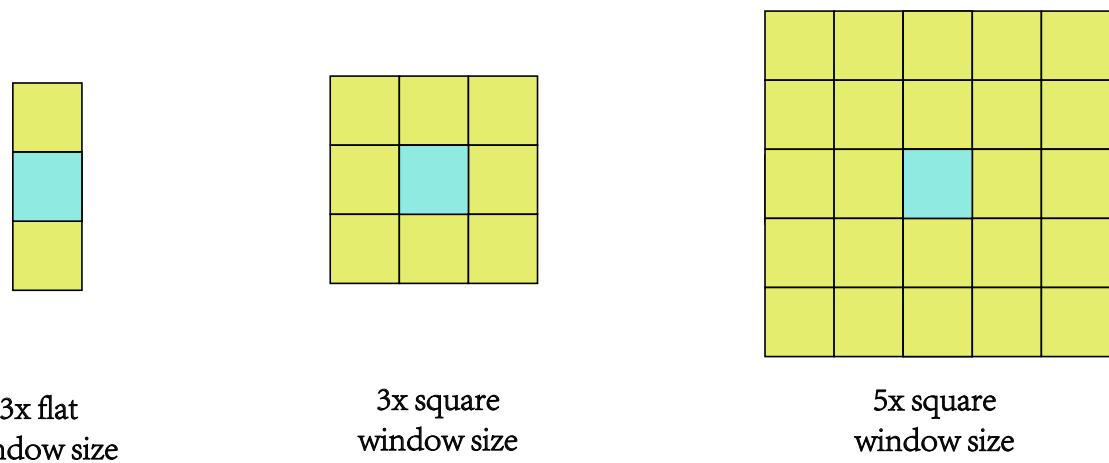


Figure 3.5: Example of variable search region

c). Multi-Pass Cross Correlation

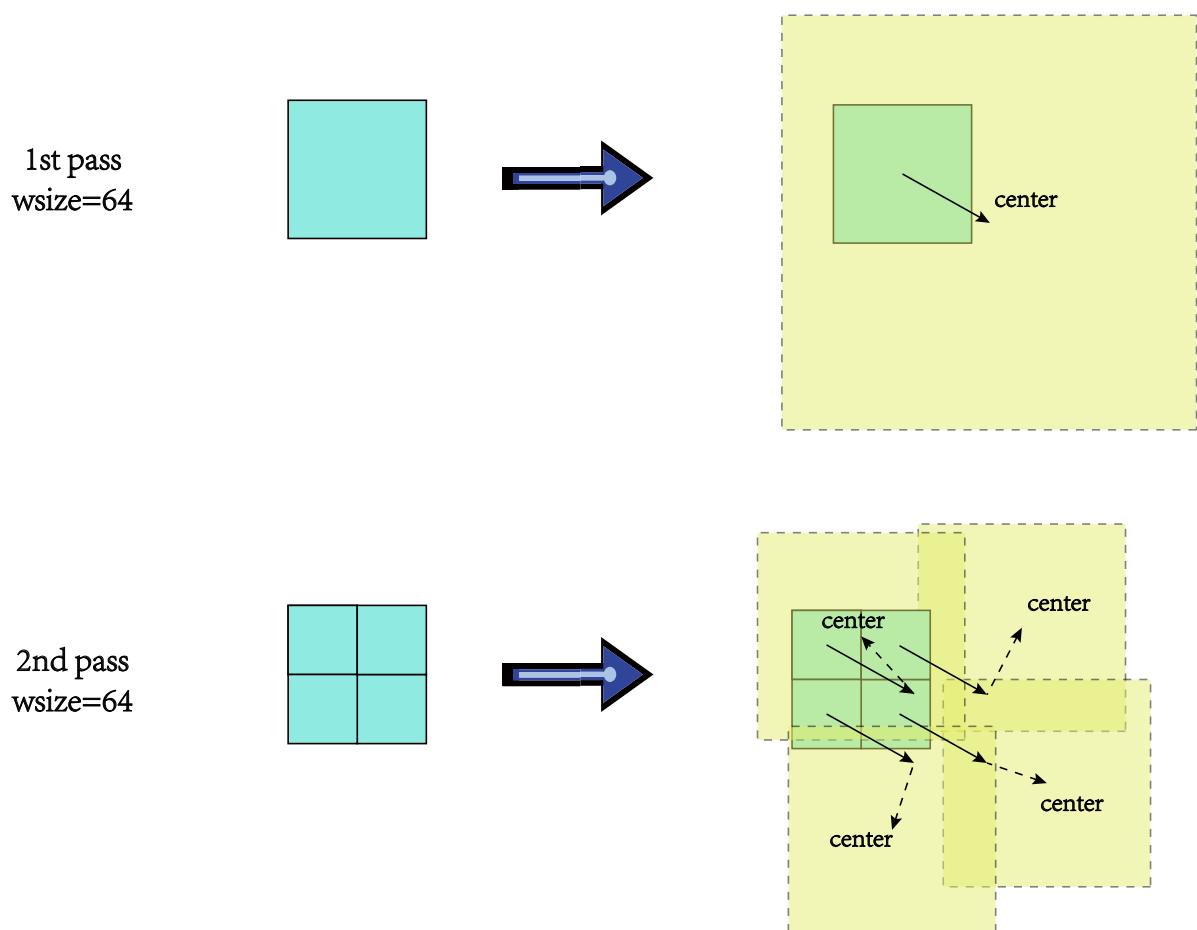


Figure 3.6: Multi pass cross correlation. (“center” in the image refer to the new search region’s center, yellow square is the new search region.)

Appendix A

Matlab code

A.1 Part 1

A.1.1 Normalised Spatial Cross Correlation in 1d

```
1 %%%%%%%%%%%%%%%%%
2 % Author: Haonan Li %
3 % Purpose: Compute the normalised cross correlation %
4 %           vector of two vectors of the same size %
5 %%%%%%%%%%%%%%%%%
6
7 function vec_c = normalised_spatial_correlation_1d(vec_a, vec_b)
8
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 % add 0 in the head and tail of vec_b
15 vec_b = [zeros(1, len-1), vec_b, zeros(1, len-1)];
16
17 % compute vec_c
18 vec_c = zeros(1, 2*len-1);
19 for i = 1:(2*len-1)
20     % nor_a, nor_b are the normalization factors
21     vec_c(i) = sum(vec_a .* (vec_b(1,i:i+len-1)))/sigma;
22 end
```

A.1.2 Signal Offset

```
1 %%%%%%%%%%%%%%%%%
2 % Author: Haonan Li %
3 % Purpose: Given two signal file, these signals have %
4 %           come from the same source and just %
5 %           offset by some time, find the offset time %
6 %%%%%%%%%%%%%%%%%
7
8 function offset = signal_offset_checker(f_1, f_2)
9
10 % read data from file
```

```

11 file_1 = importdata(f_1);
12 file_2 = importdata(f_2);
13 sig_1 = file_1.data';
14 sig_2 = file_2.data';
15 SAMPLE_RATE = 44100;
16
17 tic;
18
19 % compute cross correlation of two signal
20 % special method
21 cross_cor = spatial_correlation_1d(sig_1, sig_2)';
22 % spectral method
23 % cross_cor = spectral.correlation.function(sig_1, sig_2)';
24
25 run_time = toc
26
27 % find the position of max cross coorelation value,
28 % then compute the offset.
29 [max_value, max_pos] = max(abs(cross_cor));
30 offset = abs(length(sig_1) - max_pos)
31
32 % compute offset time and sensor distance
33 offset_time = offset / SAMPLE_RATE
34 distance = 333 * offset_time

```

A.1.3 Normalised Spatial Cross Correlation in 2d

```

1 %%%%%%
2 % Author: Haonan Li %
3 % Purpose: Receive two matrix, t(template) and A(search %
4 % region) returns normalized cross-correlation %
5 % of these two matrix, A larger than t %
6 %%%%%%
7
8 function mat_r = normalised_spatial_correlation_2d(mat_t, mat_A)
9
10 % init mat_r with 0
11 size_t = size(mat_t);
12 size_A = size(mat_A);
13 mat_r = zeros(size_A(1)-size_t(1)+1, size_A(2)-size_t(2)+1);
14 size_r = size(mat_r);
15
16 % mat_t_ is a matrix of mat_t minus mean(mat_t)
17 mat_t_ = mat_t - mean(mat_t);
18
19 for i = 1:size_r(1)
20     for j = 1:size_r(2)
21         % compute mat_r(i,j)
22         % first get the part of mat_A covered by the mat_t
23         mat_A_under = mat_A(i:(i+size_t(1)-1), j:(j+size_t(2)-1));
24         mat_A_under_ = mat_A_under - mean(mat_A_under);
25         % compute numerator and denominator of the cross correlation
26         numerator = sum(sum(mat_t_ .* mat_A_under_));
27         denominator = sqrt(sum(sum(mat_t_.^2)) * sum(sum(mat_A_under_.^2)));
28         mat_r(i,j) = numerator/denominator;
29     end
30 end

```

```

31
32 % figure
33 figure,surf(mat_r);figure(gcf)

```

A.1.4 Image Alignment

```

1 %%%%%%%%%%%%%%
2 % Author: Haonan Li %
3 % Purpose: Give two pictures, one is a part of another %
4 %           find the corresponding position and mark it %
5 %%%%%%%%%%%%%%
6
7 function pic_R = find_the_rocket_man(pic_t, pic_A)
8
9 % read images
10 pic_t = imread(pic_t);
11 pic_A = imread(pic_A);
12 mat_t = mean(pic_t, 3);
13 mat_A = mean(pic_A, 3);
14 size_t = size(mat_t);
15
16 % compute cross correlation
17 tic;
18 cross_corr = normalised_spatial_correlation_2d(mat_t,mat_A);
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+size_t(2)/2;
29 marker.YData = pos_y+size_t(1)/2;
30 marker.MarkerSize = 30;
31 marker.Color = 'r';
32 marker.MarkerFaceColor = 'r';
33 saveas(gcf, 're','png');

```

A.1.5 Spectral Cross Correlation

```

1 %%%%%%%%%%%%%%
2 % Author: Haonan Li %
3 % Purpose: Compute the spectral cross correlation %
4 %           vector of two vectors of the same size %
5 %           using Fourier transform. %
6 %%%%%%%%%%%%%%
7
8 function vec_c = spectral_correlation_1d(vec_a, vec_b)
9
10 f_a = fft(vec_a);
11 f_b = fft(vec_b);
12 conj_f_a = conj(f_a);

```

```

13 mid_res = conj_f_a .* f_b;
14 vec_c = ifft(mid_res);
15
16
17 % My fft, does not work is NN is not the power of 2
18 function res_fft = my_fft(vec)
19
20 vec_size = size(vec);
21 N = vec_size(2)
22 c = zeros(1,N);
23 % indexing computation
24 j1 = 0;
25 for i = 1 : N
26     if i < j1 + 1
27         tmp = vec(j1 + 1);
28         vec(j1 + 1) = vec(i);
29         vec(i) = tmp;
30     end
31     k = N / 2;
32     while k ≤ j1
33         j1 = j1 - k;
34         k = k / 2;
35     end
36     j1 = j1 + k;
37 end
38
39 % Butterfly computing
40 dig = 0;
41 k = N;
42 while k > 1
43     dig = dig + 1;
44     k = k / 2;
45 end
46
47 n = N / 2;
48 for m = 1 : dig
49     dist = 2 ^ (m - 1);
50     idx = 1;
51     for i = 1 : n
52         idx1 = idx;
53         for j1 = 1 : N / (2 * n)
54             r = (idx - 1) * 2 ^ (dig - m);
55             coef = exp(j * (-2 * pi * r / N));
56             tmp = 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 end
65 res_fft = vec;

```

A.1.6 Pattern Finder

```
2 % Author: Haonan Li %
3 % Purpose: Compute the spectral cross correlation %
4 % vector of two vectors of the same size %
5 % using Fourier transform. %
6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
7
8 function res = pattern-finder(wavfile,patfile)
9
10 [y1,Fs] = audioread(wavfile);
11 x1 = y1(:,1);
12
13 [y2,Fs] = audioread(wavfile);
14 x2 = y2(:,1);
15
16 corr_vec = spectral_correlation_function(x1',x2')
17
18 res = corr_vec
```

Bibliography