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

Chapter 1

Mid-term section

1.1 Introduction

Our program is a stereo vision program. Till now, we build a calibration model, which can build a 3D model through two images taken by a pair of cameras.

1.2 Outline of extension

In fact, I have great interests in artificial intelligence. I think maybe we can build a computer vision system to generate a right eye's scene from a left image. If training datasets is big enough, this should be a very easy task. But what about 3D model. Can computer vision system generate a 3D model? At least today's computer vision related research have not extend to 3D, if we can build a rudiment of it, it might be a great progress.

1.3 What I hope to learn

- For a computer science student. I want to improve my coding ability in matlab, which may gives me more oppertunities in future's job matket and gives me possibility to do computing related research.
- I want learn more about 3D vision. We live in a 3D world, I think the main part of computer vision will one day become 3D vision, so our program is a frontier research.
- The most essential part a man is ideas. I want to learn more varient way of thinking from the optimisation parts of the projects.

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-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

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

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.

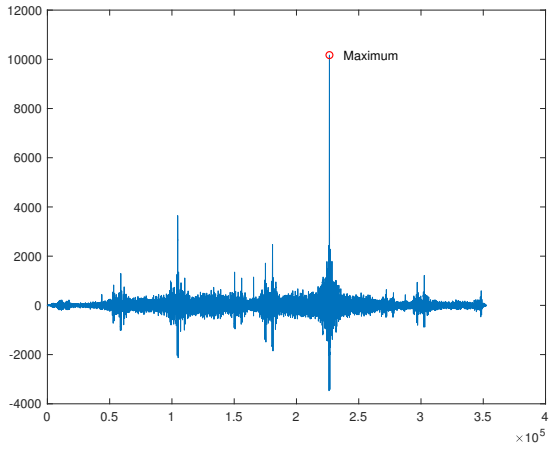


Figure 2.1: Cross correlation of two signals.

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

2.3 Normalised Spatial Cross Correlation in 2d

The cros 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

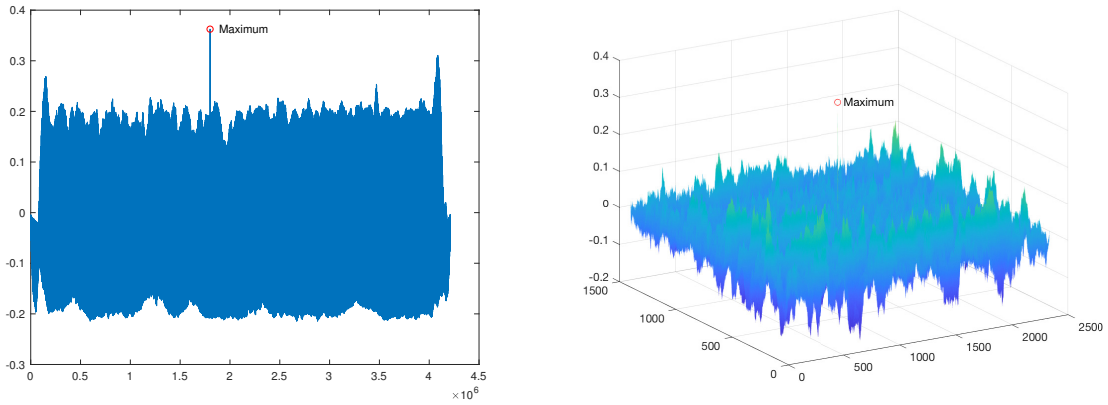
this task, we get two images, one of them is a section of the other. We can easily find where the section of the image fits in the whole through cross-correlation.

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

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

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.



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

Figure 2.3: Cross correlation result matrix visualization

Matlab code of image alognment 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 $O(n \log n)$.

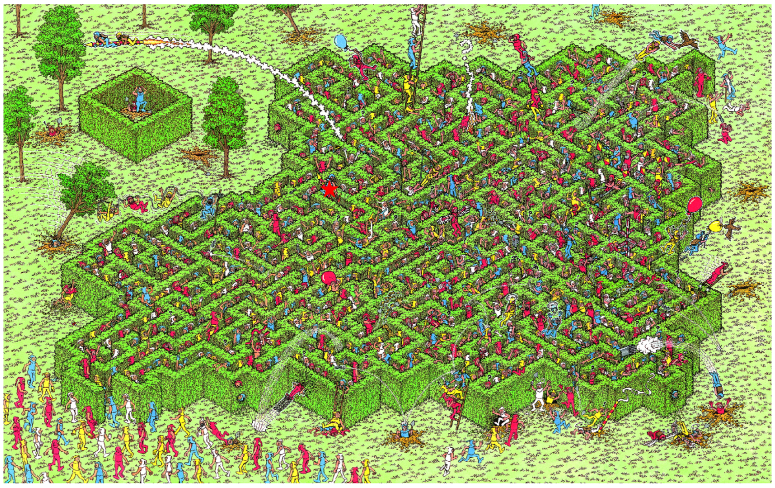


Figure 2.4: Image Alignment result.

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.

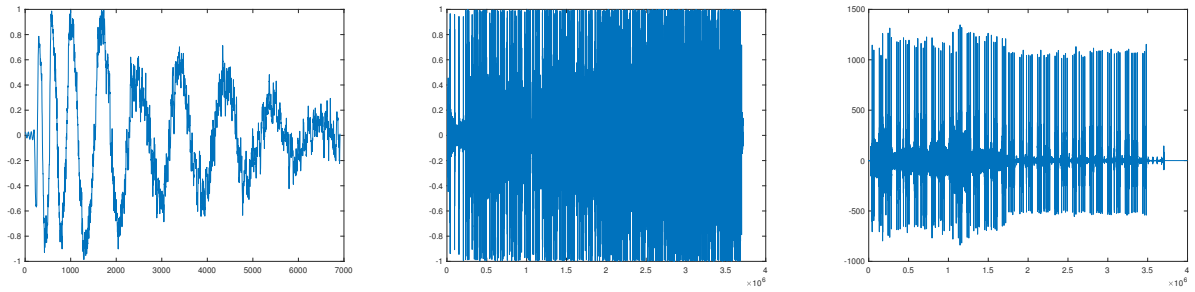
Table 2.1: Run time compare of spatial and spectral methods.

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.



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

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.

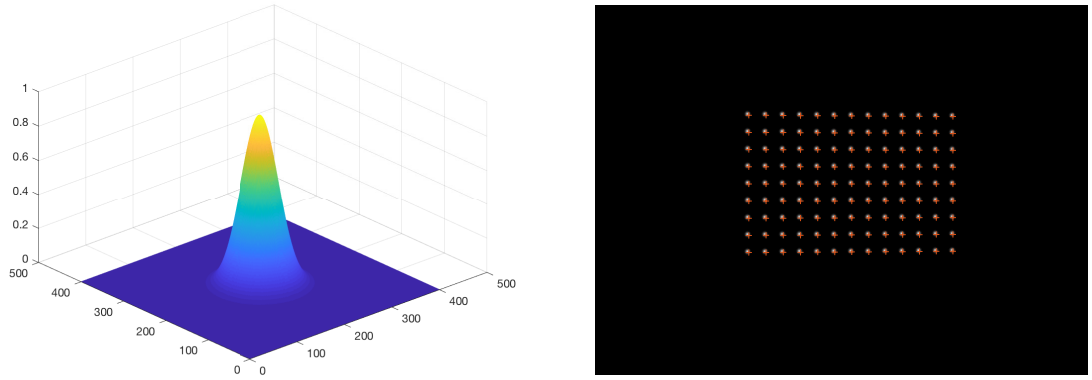
Computer vision is one of the most popular reearch today. One important reason is people believe 2d object is easy to understand and image compared with 3d object. The 2d techinology today have been more and more mature. However, we live in a 3d (even higher dimensions) world, research in 3d will become 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

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.



(a) 2D Gaussian **(b)** Detected dots

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 discribe its storage order. We sort it in order for further using shown like Figure 3.2b.

Matlab code of dot detection is in Appendix A.2.1.

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 $\pm 9°$.

(a) Disordered dots **(b)** Ordered dots

Figure 3.2: Dot Storage.

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. We finally find three funtions to fit x, y and z coordinates in real space by using corresding calibrate target's coordinates in left and right images, as

$$x_{real} = f_1(x_{left}, y_{left}, x_{right}, y_{right})$$
$$y_{real} = f_2(x_{left}, y_{left}, x_{right}, y_{right})$$
$$z_{real} = f_3(x_{left}, y_{left}, x_{right}, y_{right})$$

Matlab code of normalized 1d cross correlation is in Appendix A.2.2.

3.3 Image Comparision

After we have build calibration model. We want to apply it to practical scene. 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 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 comparision to find corresponding objects in two images and then apply it to calibration model.

We first break up one image in to windows and create a template from one window, like shown in Figure 3.3a , then create a search region (larger than 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.

Matlab code of normalized 1d cross correlation is in Appendix A.2.3.

(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).

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 optimization strategies:

- Window overlap
- Search region
- Multiple pass

a). Variable window overlap

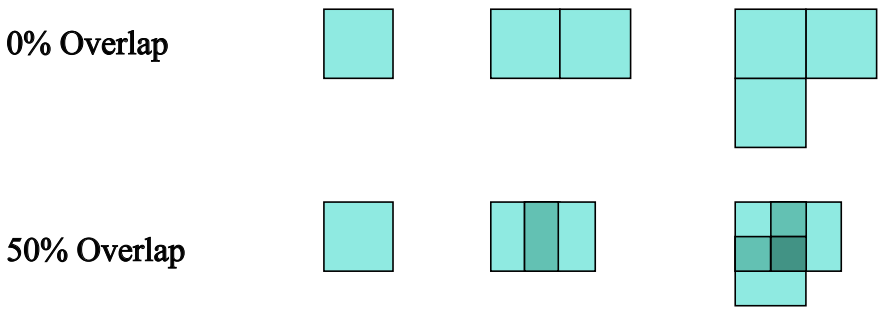


Figure 3.4: Example of variable window operlap

b).Variable search region geometry
c). Multi-Pass Cross Correlation
Matlab code of normalized 1d cross correlation is in Appendix A.2.4.

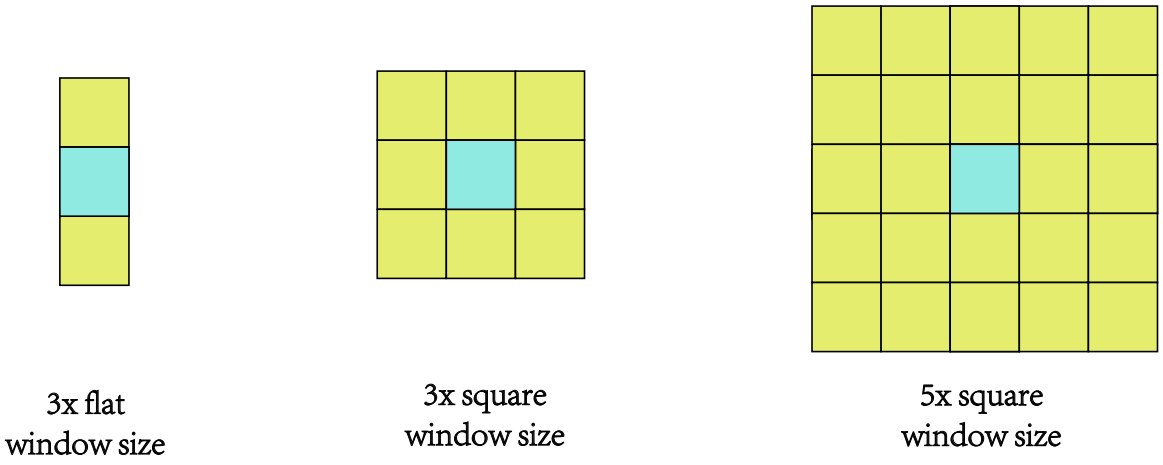


Figure 3.5: Example of variable search region

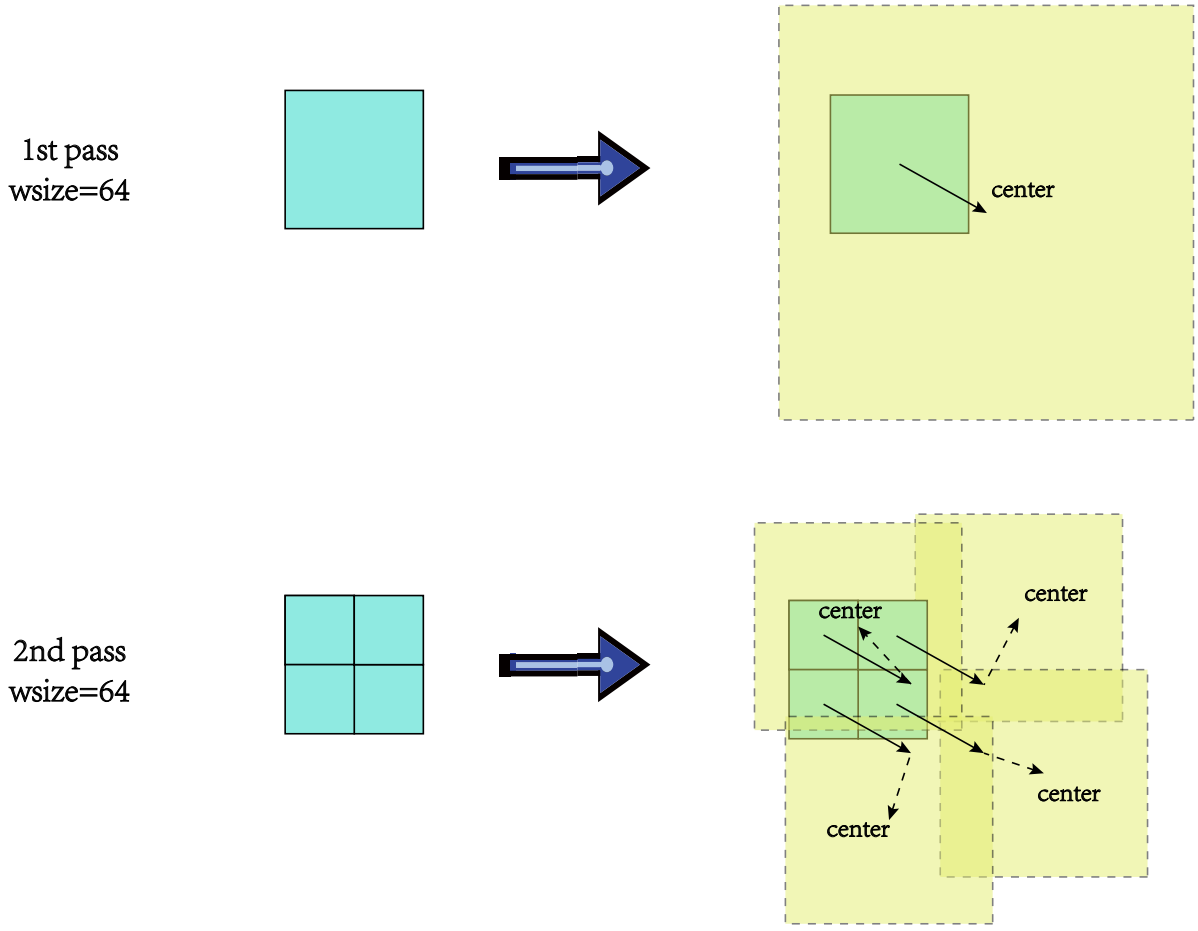


Figure 3.6: Multi pass cross correlation. ("center" in the image refer to the new search region's center, yellow square is the new search region.)

3.5 Test Scan on Computer Generated Calibrated Images

3.6 Optimised Test Scan

Appendix A

Matlab code

A.1 Part 1

A.1.1 Normalised Spatial Cross Correlation in 1d

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

function vec_c = normalised_spatial_correlation_1d(vec_a, vec_b)

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))
% add 0 in the head and tail of vec_b
vec_b = [zeros(1, len-1), vec_b, zeros(1, len-1)];

% compute vec_c
vec_c = zeros(1, 2*len-1);
for i = 1:(2*len-1)
    % nor_a, nor_b are the normalization factors
    vec_c(i) = sum(vec_a .* (vec_b(1,i:i+len-1)))/sigma;
end
```

A.1.2 Signal Offset

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author:   Haonan Li                                                    %
% Purpose:  Given two signal file, these signals have                    %
%           come from the same the same sourcce and just                 %
%           offset by some time, find the offset time                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function offset = signal_offset_checker(f_1, f_2)

% read data from file
```

```
file_1 = importdata(f_1);
file_2 = importdata(f_2);
sig_1 = file_1.data';
sig_2 = file_2.data';
SAMPLE_RATE = 44100;

tic;

% compute cross correlation of two signal
% special method
cross_cor = spatial_correlation_1d(sig_1, sig_2)';
% spectral method
% cross_cor = spectral_correlation_function(sig_1, sig_2)';

run_time = toc

% find the position of max cross coorelation value,
% then compute the offset.
[max_value, max_pos] = max(abs(cross_cor));
offset = abs(length(sig_1) - max_pos)

% compute offset time and sensor distance
offset_time = offset / SAMPLE_RATE
distance = 333 * offset_time
```

A.1.3 Normalised Spatial Cross Correlation in 2d

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author:   Haonan Li                                            %
% Purpose:  Receive two matrix, t(template) and A(search         %
%           region) returns normalized cross-correlation         %
%           of these two matrix, A larger than t                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function mat_r = normalised_spatial_correlation_2d(mat_t, mat_A)

% init mat_r with 0
size_t = size(mat_t);
size_A = size(mat_A);
mat_r = zeros(size_A(1)-size_t(1)+1, size_A(2)-size_t(2)+1);
size_r = size(mat_r);

% mat_t_ is a matrix of mat_t minus mean(mat_t)
mat_t_ = mat_t - mean(mat_t);

for i = 1:size_r(1)
    for j = 1:size_r(2)
        % compute mat_r(i,j)
        % first get the part of mat_A covered by the mat_t
        mat_A_under = mat_A(i:(i+size_t(1)-1), j:(j+size_t(2)-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_r(i,j) = numerator/denominator;
    end
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);
```

```
mid_res = conj_f_a .* f_b;
vec_c = ifft(mid_res);


% My fft, does not work is NN 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.1.6 Pattern Finder

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

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

function res = pattern_finder(wavfile,patfile)

[y1,Fs] = audioread(wavfile);
x1 = y1(:,1);

[y2,Fs] = audioread(wavfile);
x2 = y2(:,1);

corr_vec = spectral_correlation_function(x1',x2')

res = corr_vec
```

A.2 Part 2

A.2.1 Dot Detection Algorithm

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author:   Haonan Li                                          %
% Purpose:  Detect doc on a calibration plate                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function dots = dot_detect(img_t)

% read image
img = imread(img_t);
mat_img = mean(img, 3);
[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 = [-20:0.1:20];
% v = [-20:0.1:20];
% [U,V] = meshgrid(u,v);
% mat_gauss = exp(-(U.^2+V.^2)./2/3^2);
% surf(mat_gauss), shading flat
% hold on
%

cross_corr = xcorr2(mat_img, mat_gauss);
% local max cross correlation position
dots = local_max(cross_corr,50);
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 dots
% level is the local range
function res = local_max(mat_a, level)

t = 1;
res = [];
[X,Y] = size(mat_a);
for x = (1+level):(X-level)
    for y = (1+level):(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];
            end
        end
    end
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:                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function res = calibration_model()

% inpupt all data
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 dunctions
lx = left_dot(:,1);
ly = left_dot(:,2);
rx = right_dot(:,1);
ry = right_dot(:,2);
realx = real_dot(:,1);
realy = 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

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

function fitfun = myfun(capture, real)
fitfun = polyfitn(capture, real, 3);
sx      = (polyn2sym(fitfun));
terms  = fitfun.ModelTerms;
coeffs = fitfun.Coefficients ;
vars   = fitfun.VarNames;
end
```

A.2.3 Image Comparision

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author:   Haonan Li                                                   %
% Purpose:  Compare two images, split one image to                      %
%           several windows, search more similar part in                %
%           the other corresponding image                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function res = image_compare(img_a, img_b)

% read image
img_a = imread(img_a);
mat_a = mean(img_a, 3);
[a_X,a_Y] = size(mat_a);
img_b = imread(img_b);
mat_b = mean(img_b, 3);
[b_X,b_Y] = size(mat_b);

% store the result
mask_a = mat_a.*0;

```

```
res = [];
% split the larger side length to 10
wsize = round( max(a_X, a_Y) / 20);
for i = 0 : floor(a_X/wsize) - 1
    for j = 0 : floor(a_Y/wsize) - 1
        xgrid = 1 + i * wsize;
        ygrid = 1 + j * wsize;
        % find correspond points of two images
        [px,py] = corr_search(mat_a, mat_b, wsize, xgrid, ygrid);
        res = [res;xgrid,ygrid,px,py];
        dx = px - xgrid;
        dy = py - ygrid;
        mask_a(xgrid:(xgrid+wsize), ygrid:(ygrid+wsize)) = sqrt(dx*dx + ...
            dy*dy);
    end
end

% draw the corresponding points
mask_a(mask_a>10) = 0
imagesc(mask_a)
colorbar
hold on
plot(res(:,2),res(:,1),'+')
plot(res(:,4),res(:,3),'.')

end

function [px,py] = corr_search(mat_a, mat_b, wsize, xgrid, ygrid)

% pattern (window)
pat_left = xgrid;
pat_right = xgrid + wsize - 1;
pat_top = ygrid;
pat_bottom = ygrid + wsize - 1;
pattern = mat_a(pat_left:pat_right, pat_top:pat_bottom);

% search region: 3x window size square
[b_X,b_Y] = size(mat_b);
search_left = max(1, pat_left - wsize + 1);
search_right = min(b_X, pat_right + wsize);
search_top = max(1, pat_top - wsize + 1);
search_bottom = min(b_Y, pat_bottom + wsize);
search_region = mat_b(search_left:search_right, search_top:search_bottom);

% find the max cross correlation position
cross_corr = my_norm_xcorr2_2(search_region, pattern);
[rel_x,rel_y] = find(cross_corr == max(max(cross_corr)));
if isempty(rel_x)
    rel_x = zeros(1);
    rel_y = zeros(1);
end
px = rel_x(1) + search_left - 1;
py = rel_y(1) + search_top - 1;

end
```

A.2.4 Cross Correlation Optimisation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 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)

% read image
img_a = imread(img_a);
mat_a = mean(img_a, 3);
[a_X,a_Y] = size(mat_a);
img_b = imread(img_b);
mat_b = mean(img_b, 3);
[b_X,b_Y] = size(mat_b);

res = [];
% decide window size
wsize = round( min(a_X, a_Y) / 10 );
% compute shift size by overlap
shift = floor( wsize * (1 - overlap));
% image comparation
for i = 0 : floor((a_X - wsize)/shift)
    for j = 0 : floor((a_Y - wsize)/shift)
        xgrid = 1 + i * shift;
        ygrid = 1 + j * shift;
        % find correspond points of two images
        [px,py] = corr_search(mat_a, mat_b, wsize, xgrid, ygrid, sr_size, ...
            sr_shape);
        res = [res;xgrid,ygrid,px,py]
    end
end

% draw the corresponding points
imshow(img_a)
hold on
plot(res(:,2),res(:,1),'+')
plot(res(:,4),res(:,3),'.')

end


function [px,py] = corr_search(mat_a, mat_b, wsize, xgrid, ygrid, ...
    sr_size, sr_shape)

% pattern (window)
pat_left = xgrid;
pat_right = xgrid + wsize - 1;
pat_top = ygrid;
pat_bottom = ygrid + wsize - 1;
pattern = mat_a(pat_left:pat_right, pat_top:pat_bottom);

% search region
extend = floor(wsize * (sr_size - 1)/2);
[b_X,b_Y] = size(mat_b);
search_left = max(1, pat_left - extend + 1);
```

```
58 search_right = min(b_X, pat_right + extend);
59 % search region shape
60 if sr_shape == 's' % square
61     search_top = max(1, pat_top - extend + 1);
62     search_bottom = min(b_Y, pat_bottom + extend);
63 elseif sr_shape == 'f' % flat
64     search_top = pat_top
65     sear_bottom = pat_bottom
66 end
67 search_region = mat_b(search_left:search_right, search_top:search_bottom);
68
69 % find the max cross correlation position
70 cross_corr = my_norm_xcorr2_2(search_region, pattern);
71 [rel_x,rel_y] = find(cross_corr == max(max(cross_corr)));
72 % if several maxium, use the top left one
73 if isempty(rel_x)
74     rel_x = zeros(1)
75     rel_y = zeros(1)
76 end
77 px = rel_x(1) + search_left - 1;
78 py = rel_y(1) + search_top - 1;
79
80 end
```

Bibliography