Monash University: Assessment Cover Sheet

Student name	Tan	Jin Chun	
School/Campus	Monash University Malaysia	**Student's I.D. number**	32194471
Unit name	ECE4076 - Computer vision - S1 2023		
Lecturer's name	Dr. Maxine Tan	**Tutor's name**	
Assignment name	Lab 2 Submission	**Group Assignment:** No **Note, each student must attach a coversheet**	
Lab/Tute Class: Friday Lab Session	**Lab/Tute Time:** 10 a.m - 12 p.m	**Word Count:**	
Due date: 09-04-2023	**Submit Date:** 8-4-2023	**Extension granted** ☐	

If an extension of work is granted, specify date and provide the signature of the lecturer/tutor. Alternatively, attach an email printout or handwritten and signed notice from your lecturer/tutor verifying an extension has been granted.

Extension granted until (date): ..../......./........... Signature of lecturer/tutor: ....

Late submissions policy	**Days late**	**Penalty applied**
Penalties apply to late submissions and may vary between faculties. Please refer to your faculty's late assessment policy for details.		

Patient/client confidentiality: Where a patient/client case study is undertaken a signed Consent Form must be obtained.

Intentional plagiarism or collusion amounts to cheating under Part 7 of the Monash University (Council) Regulations

Plagiarism: Plagiarism means to take and use another person's ideas and or manner of expressing them and to pass these off as one's own by failing to give appropriate acknowledgement. This includes material from any source, staff, students or the Internet - published and unpublished works.

Collusion: Collusion means unauthorised collaboration on assessable written, oral or practical work with another person. Where there are reasonable grounds for believing that intentional plagiarism or collusion has occurred, this will be reported to the Associate Dean (Education) or nominee, who may disallow the work concerned by prohibiting assessment or refer the matter to the Faculty Discipline Panel for a hearing.

Student Statement:

- I have read the university's Student Academic Integrity Policy and Procedures
- I understand the consequences of engaging in plagiarism and collusion as described in Part 7 of the Monash University (Council) Regulations (academic misconduct).
- I have taken proper care to safeguard this work and made all reasonable efforts to ensure it could not be copied.
- No part of this assignment has been previously submitted as part of another unit/course.
- I acknowledge and agree that the assessor of this assignment may, for the purposes of assessment, reproduce the assignment and:

  i. provide it to another member of faculty and any external marker; and/or

  ii. submit to a text matching/originality checking software; and/or

  iii. submit it to a text matching/originality checking software which may then retain a copy of the assignment on its database for the purpose of future plagiarism checking.

- I certify that I have not plagiarised the work of others or participated in unauthorised collaboration or otherwise breached the academic integrity requirements in the Student Academic Integrity Policy.

Date: ...8./...4../..2023... Signature:...Tan Jin Chun.. *

Privacy Statement:

For information about how the University deals with your personal information go to http://privacy.monash.edu.au/guidelines/collection-personal-information.html#enrol

Assessment (ECE4076)

Lab 02 Result Document

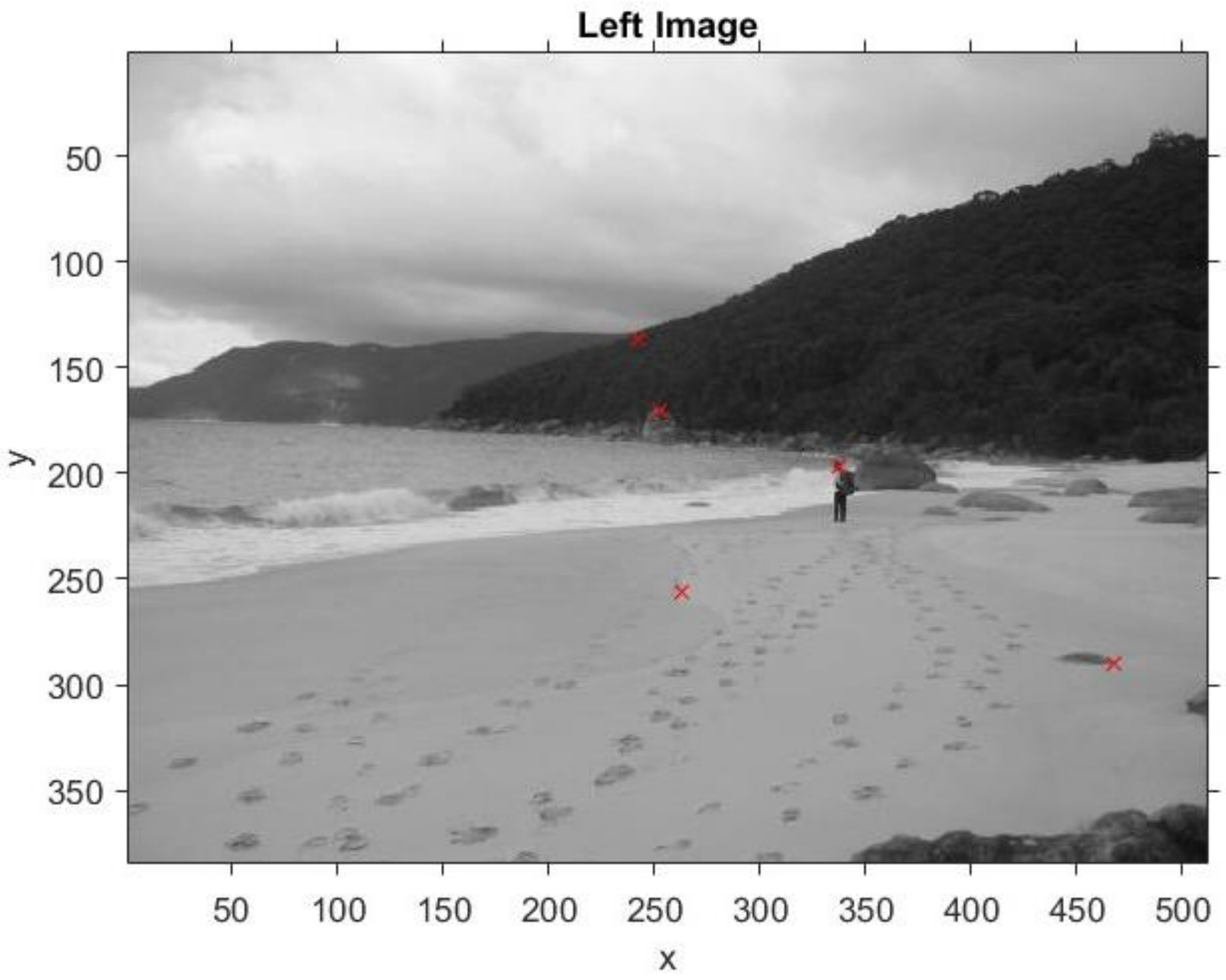
Name: Tan Jin Chun

Student ID: 32194471

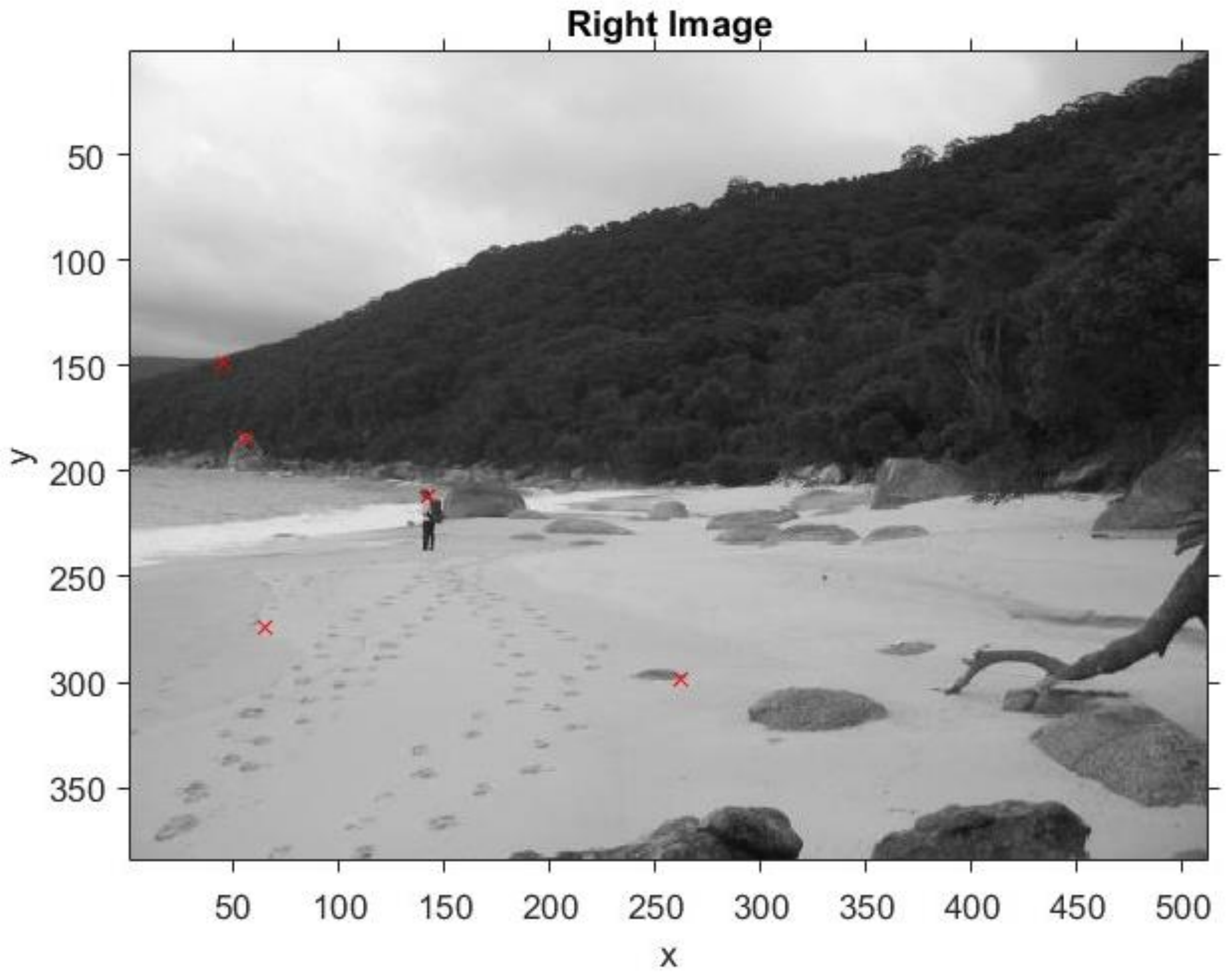
ECE4076 lab2 results document:

Name: Tan Jin Chun, Student ID: 32194471

Task 1 (1 mark):
Output for this task:



Task 2 (1 mark):
Output for this task:



Describe the process of calculating a homography matrix. Ensure you list out the key steps.

A homography matrix can be said to be a 3x3 transformation matrix that relates the coordinates of points in one image to the coordinates of the corresponding points in another image.

The process (key steps) of calculating a homography matrix is clearly stated in the lecture notes which are:
1) Obtain at least 4 matches
2) Build a matrix from the u,v,x,y values
3) Calculate the null-space of h_{ij} values
4) Repack the values into a matrix to get our homography matrix

But the steps above is not clear enough. This is more a detailed steps of the above key steps
1) We have to choose at least 4 correspondences between the two planes. These correspondences are typically selected manually or using algorithms (feature detection algorithms).
2) From here, we would have to formulate a set of linear equations based on the selected correspondence. The linear equations will relate the coordinates of the points on both of the planes. (Basically, building a matrix)
3) We then can solve the linear equations (using Gaussian Elimination or SVD (Singular Value Decomposition)) to solve the set of linear equations and obtain the homography matrix.

To further improve our homography matrix by making it more accurate, we can normalize it by dividing each element by the last element in the last row (implemented in code)

Task 3 (2 marks):
Interpolated intensity value of first transformed point (up to 2 decimal places):
66.82 (67)

Interpolated intensity value of second transformed point (up to 2 decimal places):
139.13 (140)

Interpolated intensity value of third transformed point (up to 2 decimal places):
97.96 (98)

Interpolated intensity value of fourth transformed point (up to 2 decimal places):
186.84 (187)

Interpolated intensity value of fifth transformed point (up to 2 decimal places):
58.35 (58)

Why is bilinear interpolation necessary? Which equation did you use to calculate the bilinear interpolation and why?

Bilinear Interpolation is necessary in estimating the value of a function at a point inside a rectangular grid and estimate the pixel values at non-integer coordinates.

The equation that I used to calculate the bilinear interpolation

$$f(x,y) = (1 - u)(1 - v) * f(i,j) + u(1 - v) * f(i+1,j) + (1 - u)v * f(i,j+1) + uv * f(i+1,j+1)$$

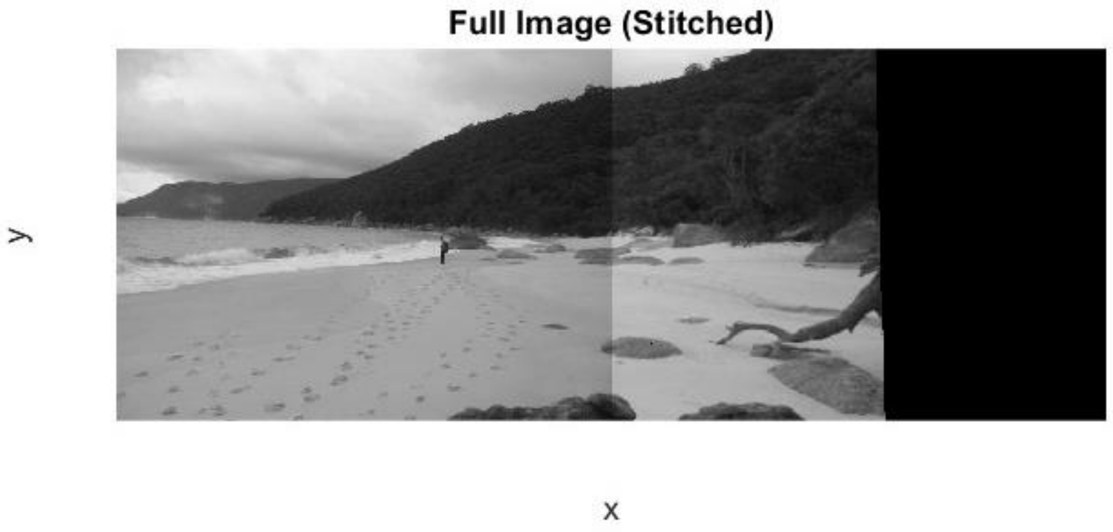
where f(x,y) is the estimated pixel value at non-integer coordinates (x,y) (estimated function value at point (x,y)), f(i,j) is the pixel value at integer coordinates (i,j) (function value at the grid point), u = x - i, and v = y - j are the fractional parts of the coordinates (fractional distances between (x,y) and the grid point (I,j)).

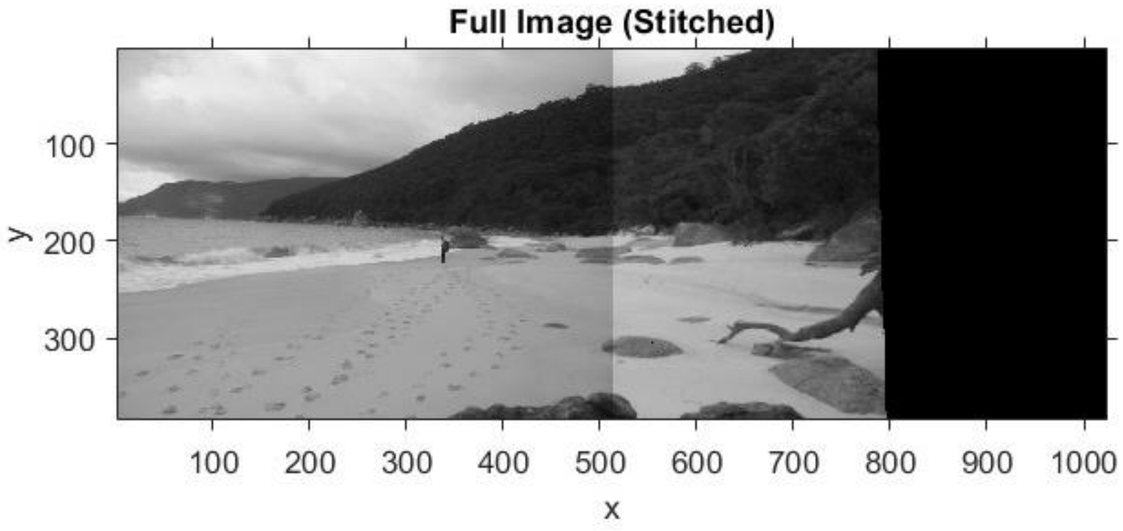
Implementing in MATLAB code

```
% interpolating in the X direction

R1 = (x2(i) - x_target(i)) / (x2(i) - x1(i)) * A + (x_target(i) -
x1(i)) / (x2(i) - x1(i)) * B;

R2 = (x2(i) - x_target(i)) / (x2(i) - x1(i)) * C + (x_target(i) -
x1(i)) / (x2(i) - x1(i)) * D;


% interpolating in the Y direction

intensity(i) = (y2(i) - y_target(i)) / (y2(i) - y1(i)) * R1 +
(y_target(i) - y1(i)) / (y2(i) - y1(i)) * R2;
```

The above equation is used because it is an efficient way to estimate the function values at non-grid points within a rectangular grid. It is also a simple and easy equation to understand.

Task 4 (2 marks):
Output for this task:





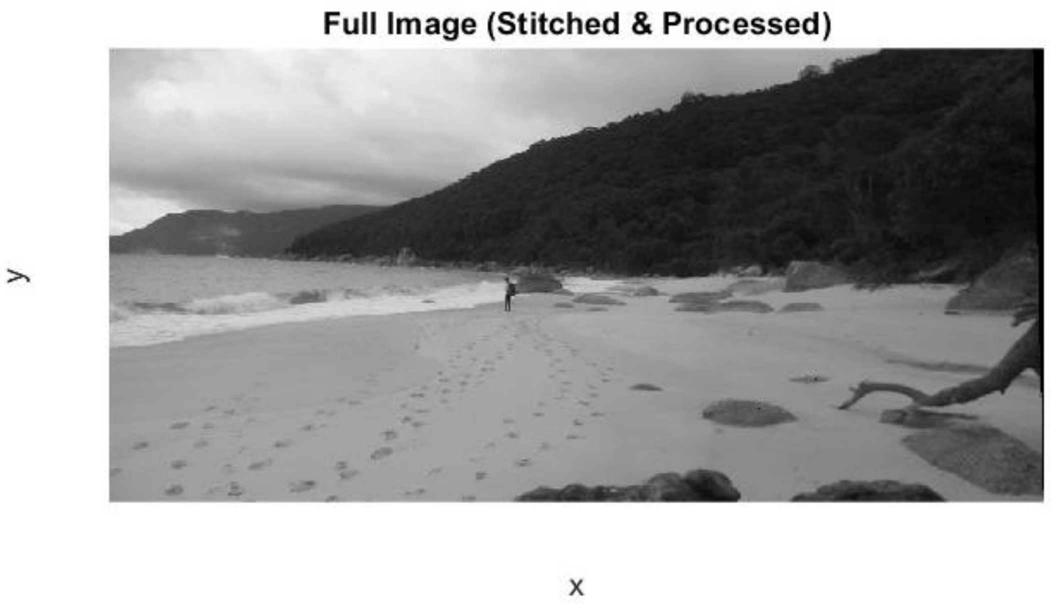
Why are some pixels invalid after applying the homography matrix?

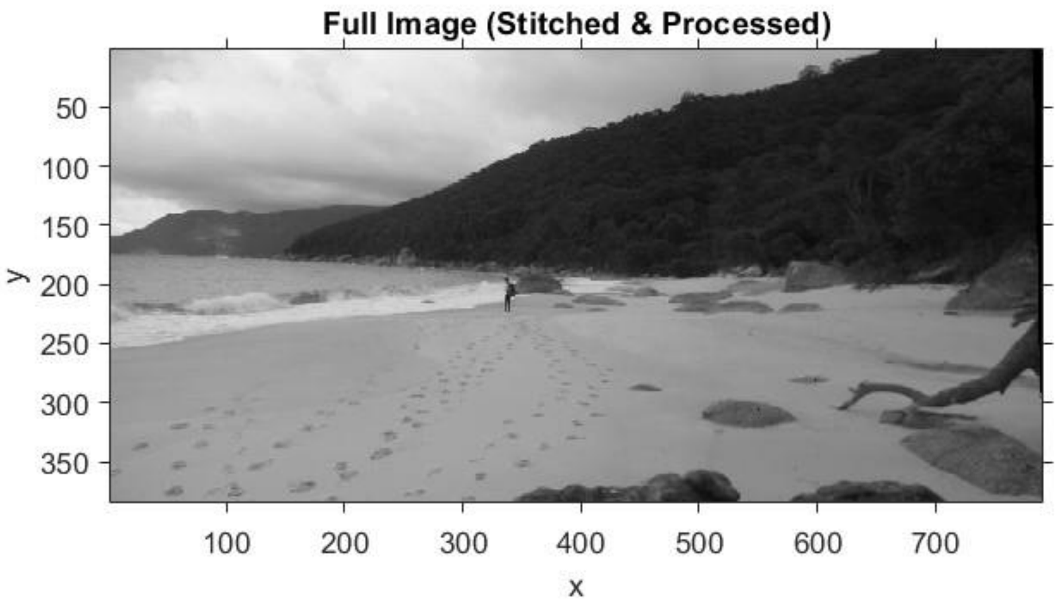
Some pixels may become invalid after applying the homography matrix as there could be discontinuities in the pixel values during the homography transformation. The homography transformation can also cause a point in the source image to be mapped to a location which is outside the bounds of the destination image. This would cause the pixel values to be undefined/invalid. The homography transformation can also cause some pixels in the source image to be hidden from view in the destination image, rendering the pixel values at the hidden location in the destination image to be invalid. The interpolation of the neighbouring pixel values in the source image can cause some pixels in the destination image to be invalid.

In a nutshell, the reasons why some pixels become invalid after applying the homography matrix can be summed up as follows:
1) Discontinuities in the pixel values
2) Point in the source image located out of bounds of the destination image
3) Hidden pixels in the source image from view in the destination image
4) Interpolation of the pixel values would cause the estimated pixel values to be invalid

We can rectify the problem above by using smoother homography transformation to minimize discontinuities and cropping our destination image to exclude out of bounds regions.

Task 5 (2 marks):
Output for this task:





Describe the steps you have used to improve the blending process. Why were they effective?

There were several steps that I have used to improve the blending process. The first step was that I cropped the input image to remove any unnecessary areas and focus only on the relevant parts. This will help to reduce the size of the images which will remove any distortion or misalignments in the input images. Basically, just adjusting the width of the output image so that less black pixels are visible.

The next step that I did was that I adjusted the brightness of each image (by a scaling factor) so that the seam is less visible. This is like somewhat creating a mask by defining the brightness of part of the image so that we can blend parts of the image together to make the image much more precise.

I have also adjusted the horizontal location of the seam (can be moved further to the left as the right image overlap the left image by a few pixels).

The last step was that I applied a small amount of Gaussian Blur / alpha blending near the seam to make it less visible. Applying a Gaussian Blur to the mask can help to smooth out the sharp edges and transitions between the two images, making the image much more natural looking.

These steps can be effective in improving the blending process because they help reduce the visual differences between the two images, localize the blending process to specific areas, smooth out sharp edges and transitions, and create a seamless and natural-looking blend between the two images.

Task 6 (Optional):
Output for this task:

Code for Task 1:

Paste your code in here.

```
% Written by Nigel Tan Jin Chun
% Last Modified: 4/4/2023
% Name of the file: Lab2_Task1
% Function of the file: Drawing the stated points onto the left
image as
% red crosses

clear all;clc;close all;

% Declaring and intialising the variable
% Using imread to read the given left image
left_image = imread("left.jpg");

% The stated positions
stated_points = [338,197,1;
                 468,290,1;
                 253,170,1;
                 263,256,1;
                 242,136,1];

% Getting the 2-D Red Crosses by transforming the 3-element
homogenous
% coordinates to 2D image pixel coordinates by dividing the first
and
% second elements by the third elements
red_crosses = [stated_points(:,1) ./ stated_points(:,3),
stated_points(:,2) ./ stated_points(:,3)];

% Plotting the red crosses
figure(1);
imshow(left_image, 'InitialMagnification','fit');
hold on;
axis on;
for i = 1:5
    plot(red_crosses(i,1), red_crosses(i,2), 'rx');
end
hold off;
title("Left Image");
xlabel("x");
ylabel("y");
```

Code for Task 2:

Paste your code in here.

```
% Written by Nigel Tan Jin Chun
% Last Modified: 4/4/2023
% Name of the file: Lab2_Task2
% Function of the file: Drawing the stated points onto the left
image as
% red crosses

clear all;clc;close all;

% Declaring and intialising the variable
% Using imread to read the given right image
right_image = imread("right.jpg");

% The stated positions
stated_points = [338,197,1;
                 468,290,1;
                 253,170,1;
                 263,256,1;
                 242,136,1];

% The homography
homography = [1.6010, -0.0300, -317.9341;
              0.1279, 1.5325,  -22.5847;
              0.0007, 0,       1.2865];

% The transformed points
% Applying the homography to transform the left image to the
corresponding
% locations in the right image
transformed_points = (homography * stated_points')';

% Getting the 2-D Red Crosses by transforming the 3-element
homogenous
% coordinates to 2D image pixel coordinates by dividing the first
and
% second elements by the third elements
red_crosses = [transformed_points(:,1) ./ transformed_points(:,3),
transformed_points(:,2) ./ transformed_points(:,3)];

% Plotting the red crosses
figure(1);
imshow(right_image, 'InitialMagnification','fit');
hold on;
axis on;
for i = 1:5
    plot(red_crosses(i,1), red_crosses(i,2), 'rx');
end
hold off;
title("Right Image");
xlabel("x");
ylabel("y");
```

Code for Task 3:

Paste your code in here.

Function File Created

```
% Written by Nigel Tan Jin Chun
% Last Modified: 4/4/2023
% Name of the file: myInterp2.m
% Function of the file: Take the transformed pixel coordinate and
the
% intensity values of its four neighbours as input arguments and
output the
% interpolated intensity values


function intensity = myInterp2(x_target, y_target, image)

    % Determine the nearest integer pixel coordinates
    x1 = floor(x_target);
    x2 = ceil(x_target);
    y1 = floor(y_target);
    y2 = ceil(y_target);

    % Convert the pixel values to floating-point numbers for the
bilinear interpolation computation
    % image = double(image);

    % Initialising the variables for efficiency allocation purpose
    intensity = zeros(1,length(x1));

    % Using a for loop to calculate the interpolated intensity value
    for i = 1:length(intensity)

        % A, B, C, D are the intensity of the neighboring pixels
        A = (image(y1(i), x1(i)));
        B = (image(y1(i), x2(i)));
        C = (image(y2(i), x1(i)));
        D = (image(y2(i), x2(i)));

        % interpolating in the X direction
        R1 = (x2(i) - x_target(i)) / (x2(i) - x1(i)) * A +
(x_target(i) - x1(i)) / (x2(i) - x1(i)) * B;
        R2 = (x2(i) - x_target(i)) / (x2(i) - x1(i)) * C +
(x_target(i) - x1(i)) / (x2(i) - x1(i)) * D;

        % interpolating in the Y direction
        intensity(i) = (y2(i) - y_target(i)) / (y2(i) - y1(i)) * R1
+ (y_target(i) - y1(i)) / (y2(i) - y1(i)) * R2;
    end
end
```

```
% Written by Nigel Tan Jin Chun
% Last Modified: 4/4/2023
% Name of the file: Lab2_Task3
% Function of the file: Computing the intensity of the transformed
pixel
% coordinate in the right.jpg

clear all;clc;close all;

% Declaring and intialising the variable
% Using imread to read the given right image
right_image = imread("right.jpg");

% The stated positions
stated_points = [338,197,1;
                 468,290,1;
                 253,170,1;
                 263,256,1;
                 242,136,1];

% The homography
homography = [1.6010, -0.0300, -317.9341;
              0.1279, 1.5325,  -22.5847;
              0.0007, 0,       1.2865];

% The transformed points
% Applying the homography to transform the left image to the
corresponding
% locations in the right image
transformed_points = (homography * stated_points')';

% Getting the 2-D Red Crosses by transforming the 3-element
homogenous
% coordinates to 2D image pixel coordinates by dividing the first
and
% second elements by the third elements
red_crosses = [transformed_points(:,1) ./ transformed_points(:,3),
transformed_points(:,2) ./ transformed_points(:,3)];

% Calling the written function
intensity = myInterp2(red_crosses(:,1), red_crosses(:,2),
right_image);

% Comparing with the built-in function
% check_intensity = interp2(x, y, z, xi, yi, 'linear');

% Displaying the interpolated intensity value
display(intensity);
```

Code for Task 4:

Paste your code in here.

```
% Written by Nigel Tan Jin Chun
% Last Modified: 4/4/2023
% Name of the file: Lab2_Task4
% Function of the file: Basically stiching the two images together

clear all;clc;close all;

% Declaring and intialising the variable
% Using imread to read the given left and right image
left_image = imread("left.jpg");
right_image = imread("right.jpg");

% Creating the 1024x384 image
new_image = zeros(384,1024);

% The homography
homography = [1.6010, -0.0300, -317.9341;
              0.1279, 1.5325,  -22.5847;
              0.0007, 0,       1.2865];

% Convert the pixel values of the left image to floating-point
numbers
double_left_image = double(left_image);

% Getting the dimensions of the image
[height_left_image, width_left_image] = size(left_image);

% Filling the LHS with the left image
new_image(:, 1:width_left_image) = double_left_image;

% Setting up the coordinate vector with z = 1
% Filling in the remaining 512x384 pixels on the RHS
empty_grid = zeros(384*512,3);

% Initialising the variable for looping
x = 512 + 1;
y = 1;

% Using the for loop to fill up the empty grid with 3-element
homogeneous
% coordinates
for i = 1:(width_left_image*height_left_image)
    empty_grid(i,:) = [x, y, 1];
    y = y + 1;

    if (y > height_left_image)
        y = 1;
        x = x + 1;
    end

end
```

```
% If the right pixel coordinate is valid, generate the pixel value
using
% the bilinear interpolation function that we have created
(myInterp2)

% If the right pixel coordinate is invalid, use a pixel value of 0

% For each pixel in empty_grid, the homography matrix H is used to
transform the
% pixel coordinates from the right image to the left image.
% The transformed coordinates are then used to perform bilinear
interpolation
% on the right input image using the bilinearInterpolation function.

% Declaring and initialising the variable
% transformed coordinates z = 1
rc = [];

% transformed points
transformed_points = 0;
intensity_points = 0;

% pixel by pixel calculation of intensity
for i=1:length(empty_grid)

    % Applying the homography to transform the left image to the
corresponding
    % locations in the right image
    transformed_points = (homography * empty_grid(i,:)')';

    % transforming the 3-element homogenous
    % coordinates to 2D image pixel coordinates by dividing the
first and
    % second elements by the third elements
    rc = [transformed_points(1)./transformed_points(3),
transformed_points(2)./transformed_points(3)];

    if rc(1) > 512
        continue
    end

    % Calling the written function
    intensity_points = myInterp2(rc(1), rc(2), right_image);

    % Filling the new values (newly found intensity values) into the
new
    % image
    new_image(empty_grid(i,2), empty_grid(i,1)) = intensity_points;
end

% Displaying the newly formed image
figure(3);
imshow(new_image, [0 255],'InitialMagnification','fit');
title("Full Image (Stitched)");
xlabel("x");
```

```
ylabel("y");

figure(4);
imshow(new_image, [0 255],'InitialMagnification','fit');
axis on;
title("Full Image (Stitched)");
xlabel("x");
ylabel("y");
```

Code for Task 5:

Paste your code in here.

```
% Written by Nigel Tan Jin Chun
% Last Modified: 4/4/2023
% Name of the file: Lab2_Task5
% Function of the file: Improving the visual quality of the stiched
image

% The main 4 image processing techniques
% Adjusting the width of output image so that less black pixels are
visible
% Adjusting the brightness (by a scalling factor) of each image so
that the
% seam is less visible
% Applying a small amount of Gaussian Blur of Alpha bending near the
seam
% to make it less visible
% Adjusting the horizontal location of the seam (it can be moved
further to the left
% as the right image)

clear all;clc;close all;

% Declaring and intialising the variable
% Using imread to read the given left and right image
left_image = imread("left.jpg");
right_image = imread("right.jpg");

% Creating the 1024x384 image
new_image = zeros(384,1024);

% The homography
homography = [1.6010, -0.0300, -317.9341;
              0.1279, 1.5325,  -22.5847;
              0.0007, 0,       1.2865];

% Convert the pixel values of the left image to floating-point
numbers
double_left_image = double(left_image);

% Getting the dimensions of the image
[height_left_image, width_left_image] = size(left_image);

% Filling the LHS with the left image
new_image(:, 1:width_left_image) = double_left_image;

% Setting up the coordinate vector with z = 1
% Filling in the remaining 512x384 pixels on the RHS
empty_grid = zeros(384*512,3);

% Initialising the variable for looping
x = 512 + 1;
y = 1;
```

```
% Using the for loop to fill up the empty grid with 3-element
homogeneous
% coordinates
for i = 1:(width_left_image*height_left_image)
    empty_grid(i,:) = [x, y, 1];
    y = y + 1;

    if (y > height_left_image)
        y = 1;
        x = x + 1;
    end

end

% If the right pixel coordinate is valid, generate the pixel value
using
% the bilinear interpolation function that we have created
(myInterp2)

% If the right pixel coordinate is invalid, use a pixel value of 0

% For each pixel in empty_grid, the homography matrix H is used to
transform the
% pixel coordinates from the right image to the left image.
% The transformed coordinates are then used to perform bilinear
interpolation
% on the right input image using the bilinearInterpolation function.

% Declaring and initialising the variable
% transformed coordinates z = 1
rc = [];

% transformed points
transformed_points = 0;
intensity_points = 0;

% Declaring the smallest x and largest x variable
largest_x = 0;
smallest_x = 1024;

% pixel by pixel calculation of intensity
for i=1:length(empty_grid)

    % Applying the homography to transform the left image to the
corresponding
    % locations in the right image
    transformed_points = (homography * empty_grid(i,:)')';

    % transforming the 3-element homogenous
    % coordinates to 2D image pixel coordinates by dividing the
first and
    % second elements by the third elements
    rc = [transformed_points(1)./transformed_points(3),
transformed_points(2)./transformed_points(3)];
```

```
    if rc(1) > 512
        continue
    end

    % Adding extra conditions to check for here
    if empty_grid(i,1) > largest_x
       largest_x = empty_grid(i,1);
    end
    if floor(rc(1)) < smallest_x
       smallest_x = floor(rc(1));
    end

    % Calling the written function
    intensity_points = myInterp2(rc(1), rc(2), right_image);

    % Filling the new values (newly found intensity values) into the
new
    % image
    new_image(empty_grid(i,2), empty_grid(i,1)) = intensity_points;

end

% Cropping of image
% Automatic cropping of image
new_image = new_image(:, 1:largest_x);

% Declaring and initialising new variables
% Scalling of brightness
crop_right_image = uint8(new_image(:, 513:largest_x));

% Window-by-Window Brightness Adjustment
processed_crop_right_image = crop_right_image;
processed_crop_right_image(1:50,:) =
processed_crop_right_image(1:50, :)*0.75;
processed_crop_right_image(51:100,:) =
processed_crop_right_image(51:100, :)*0.72;
processed_crop_right_image(101:150,:) =
processed_crop_right_image(101:150, :)*0.72;
processed_crop_right_image(151:200,:) =
processed_crop_right_image(151:200, :)*0.805;
processed_crop_right_image(201:300,:) =
processed_crop_right_image(201:300, :)*0.805;
processed_crop_right_image(301:320,:) =
processed_crop_right_image(301:320, :)*0.805;
processed_crop_right_image(321:340,:) =
processed_crop_right_image(321:340, :)*0.78;
processed_crop_right_image(341:360,:) =
processed_crop_right_image(341:360, :)*0.775;
processed_crop_right_image(361:384,:) =
processed_crop_right_image(361:384, :)*0.775;

% Gaussian blur at the seam and brightness control at the stitched
RHS
processed_stitch_image = [uint8(new_image(:, 1:512)),
processed_crop_right_image(:,5:end)];
```

```
processed_stitch_image(221:end, 470:520) =
imgaussfilt(double(processed_stitch_image(221:end, 470:520)),
'FilterSize', [21 21]);
processed_stitch_image(361:370, 470:520) =
imgaussfilt(double(processed_stitch_image(361:370, 470:520)),
'FilterSize', [5 5]);
processed_stitch_image(361:370, 470:520) =
imgaussfilt(double(processed_stitch_image(361:370, 470:520)),
'FilterSize', [5 5]);
processed_stitch_image(321:340, 470:520) =
imgaussfilt(double(processed_stitch_image(321:340, 470:520)),
'FilterSize', [5 5]);

% Displaying the newly formed stitched and processed image
figure(5);
imshow(processed_stitch_image, [0 255], 'InitialMagnification',
'fit');
title("Full Image (Stitched & Processed)");
xlabel("x");
ylabel("y");

figure(6);
imshow(processed_stitch_image, [0 255], 'InitialMagnification',
'fit');
axis on;
title("Full Image (Stitched & Processed)");
xlabel("x");
ylabel("y");
```

Code for Task 6:

```
Paste your code in here.
```