# 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 1 Results Document Submission | | **Group Assignment:** No<br>**Note, each student must attach a coversheet** | |
| **Lab/Tute Class:** Friday Lab Session | | **Lab/Tute Time:** 10.00 am - 12.00 pm | | **Word Count:** |
| **Due date**: 26-03-2023 | | **Submit Date:** 26-03-2023 | | **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 This includes material from any source, staff, students or the Internet - published and unpublished works.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: 26 / 03 / 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 01 Result Document

Name: Tan Jin Chun
Student ID: 32194471

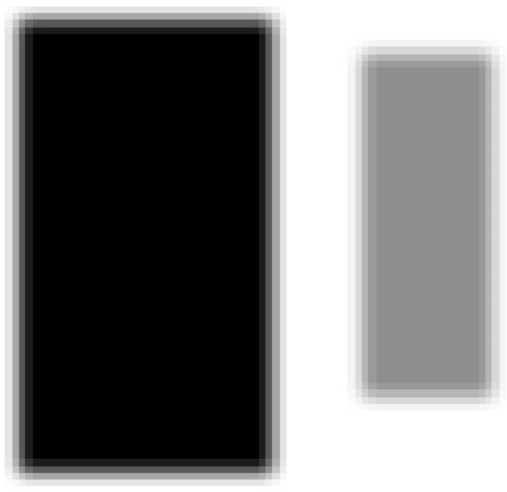
# ECE4076 lab1 results document:

Name: Nigel Tan Jin Chun, Student ID: 32194471

**Task 1 (2 marks):**

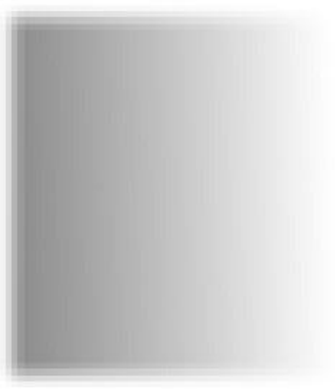
*Output for this task with test00 image:*

Test 00 Gaussian Blur Image

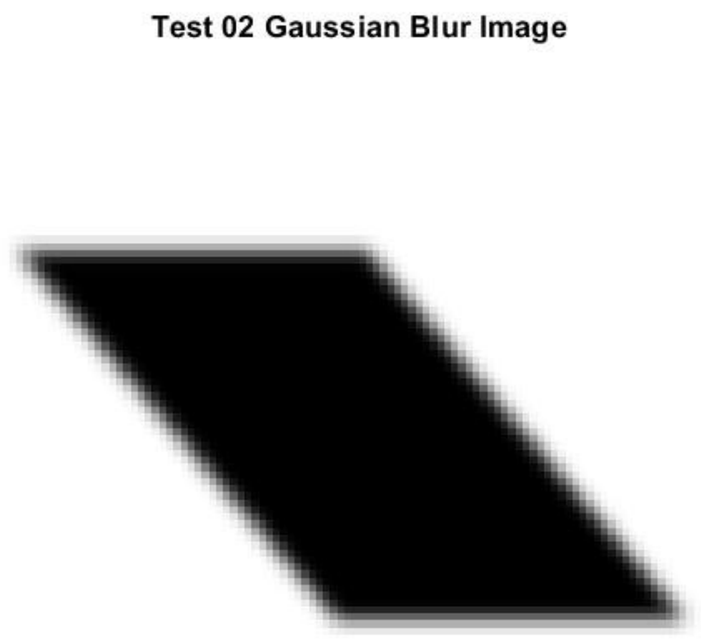


*Output for this task with test01 image:*

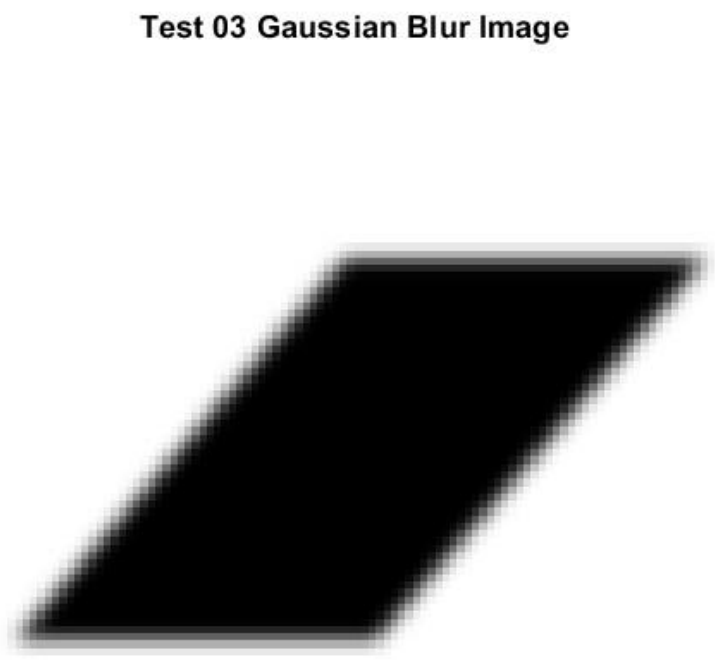
Test 01 Gaussian Blur Image



*Output for this task with test02 image:*

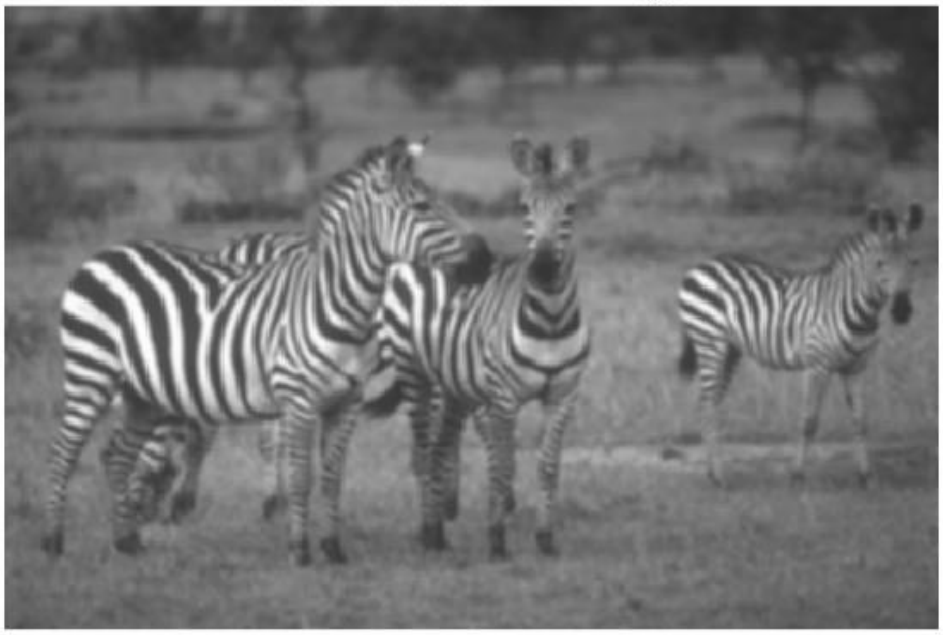


*Output for this task with test03 image:*



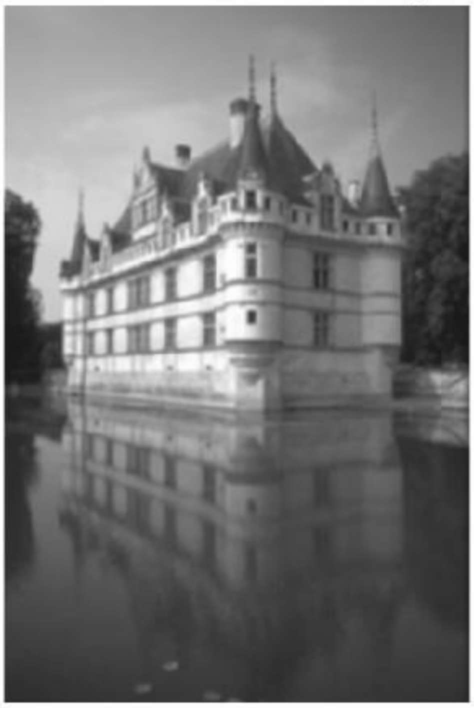
*Output for this task with test04 image:*

Test 04 Gaussian Blur Image

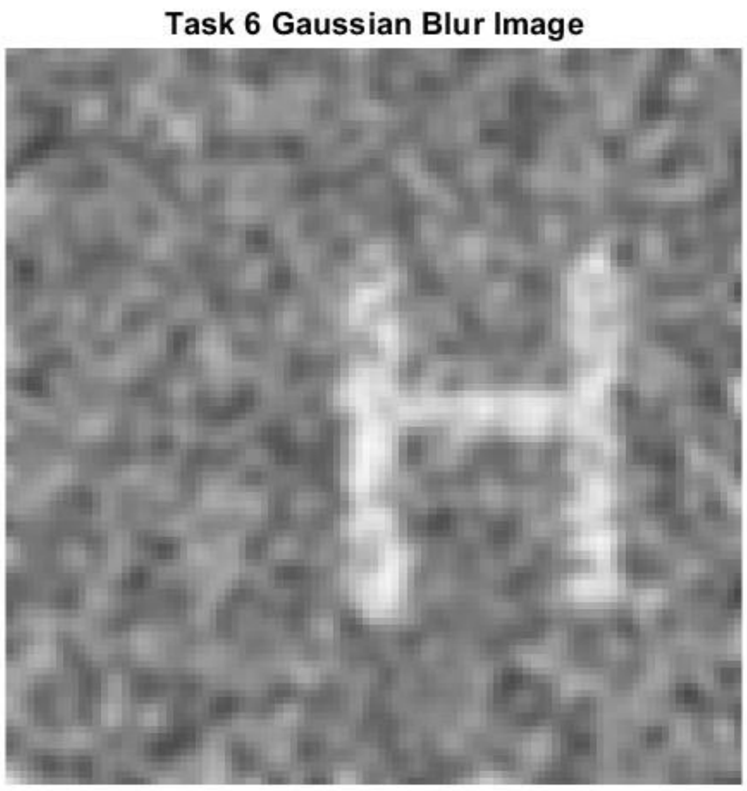


*Output for this task with test05 image:*

Test 05 Gaussian Blur Image



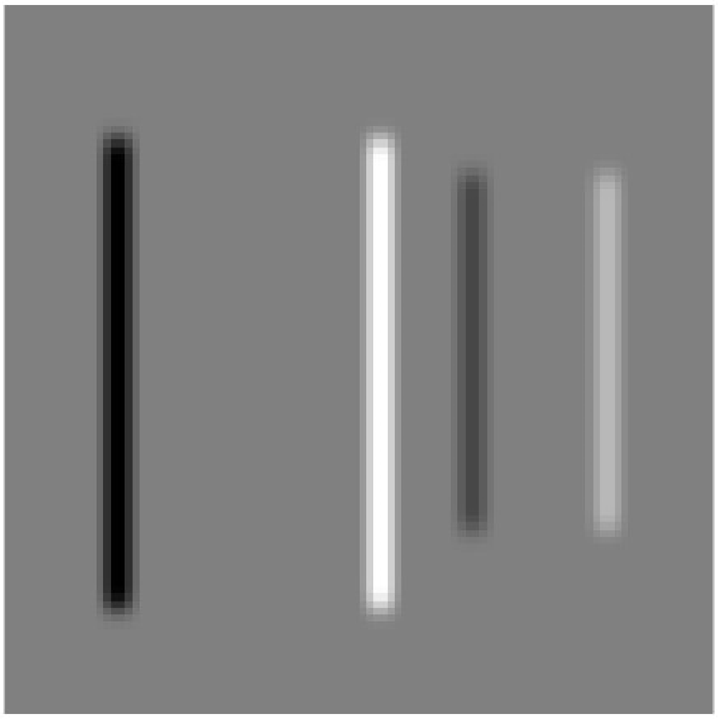
*Output for this task with task06 image:*



**Task 2 (1 mark):**

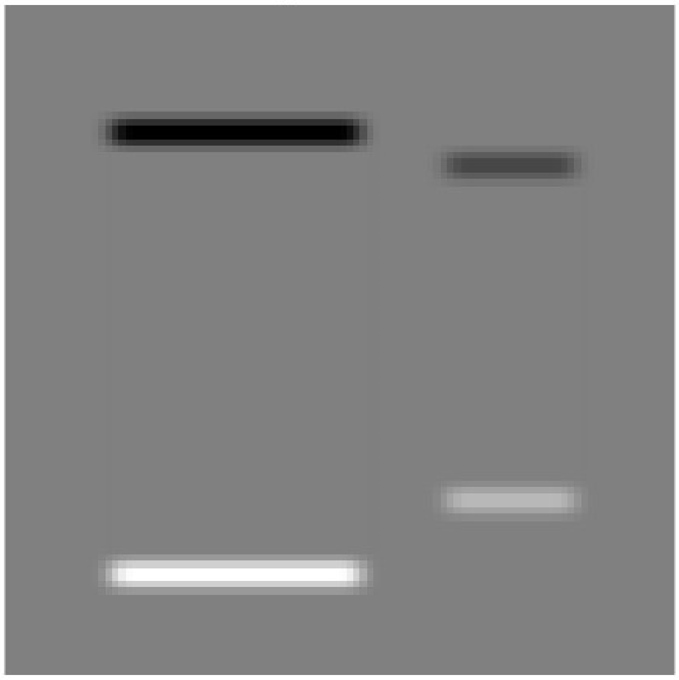
*Output of Gx image with test00 image:*

**Gx of test00**



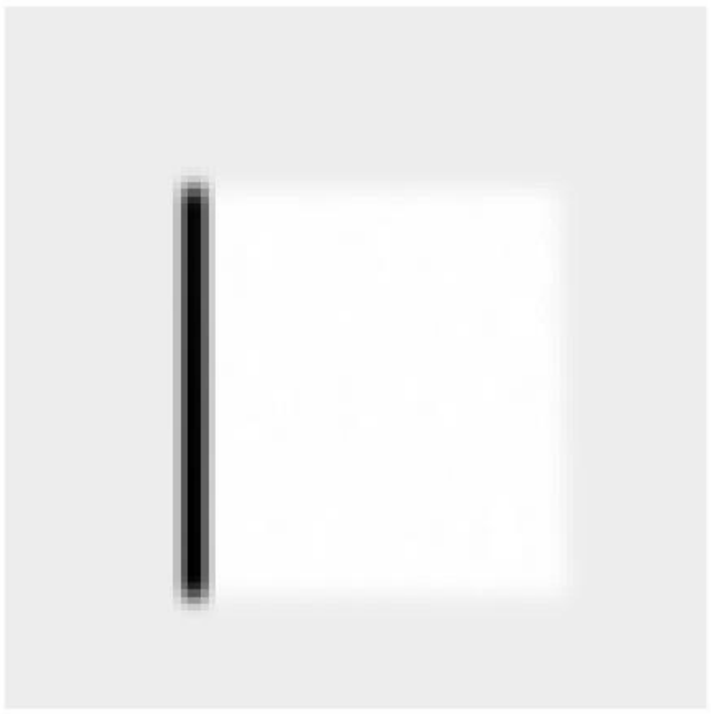
*Output of Gy image with test00 image:*

**Gy of test00**



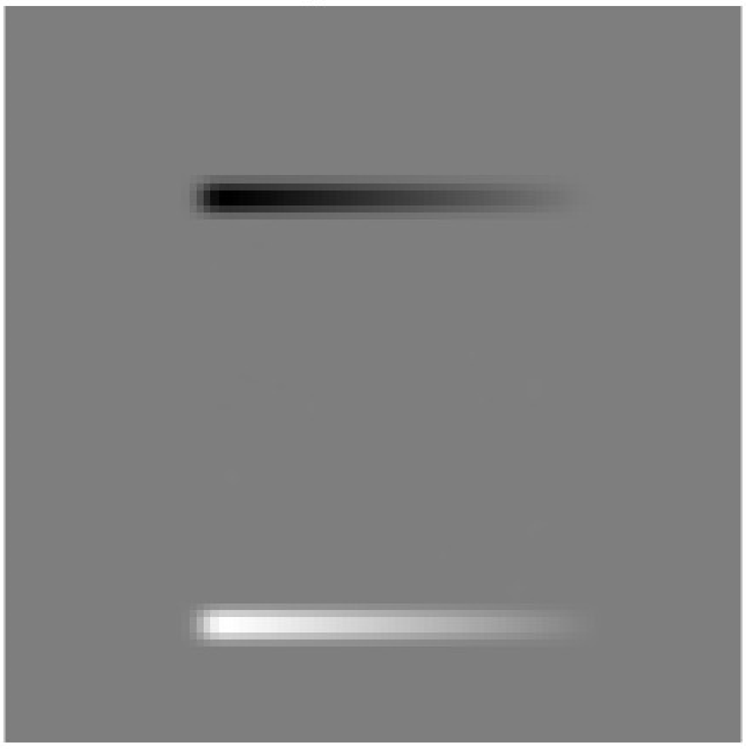
*Output of Gx image with test01 image:*

Gx of test01

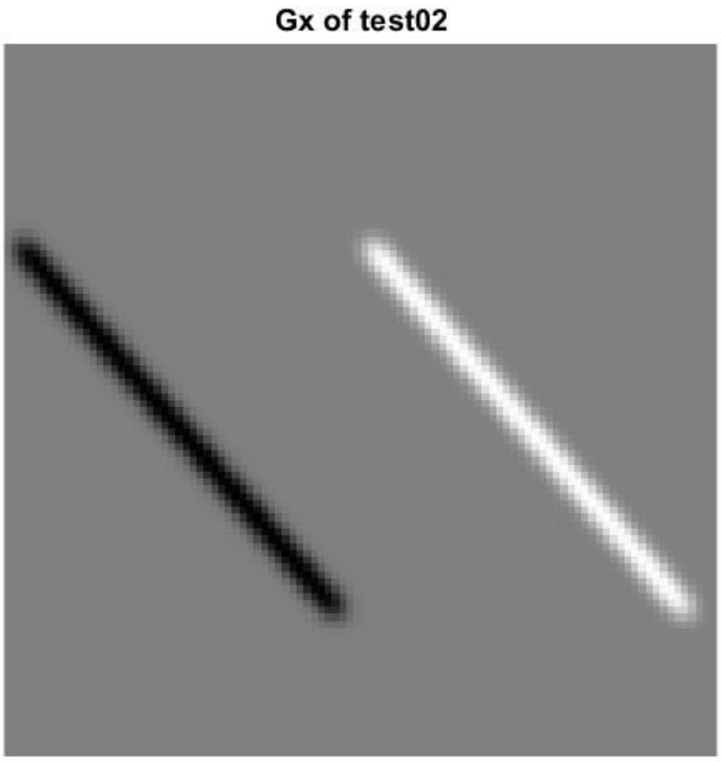


*Output of Gy image with test01 image:*

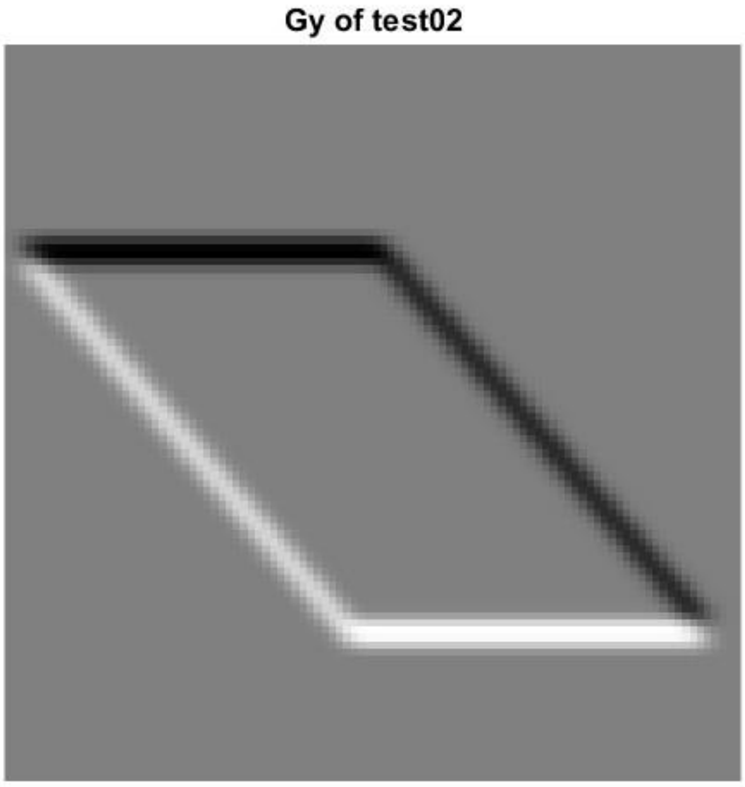
Gy of test01



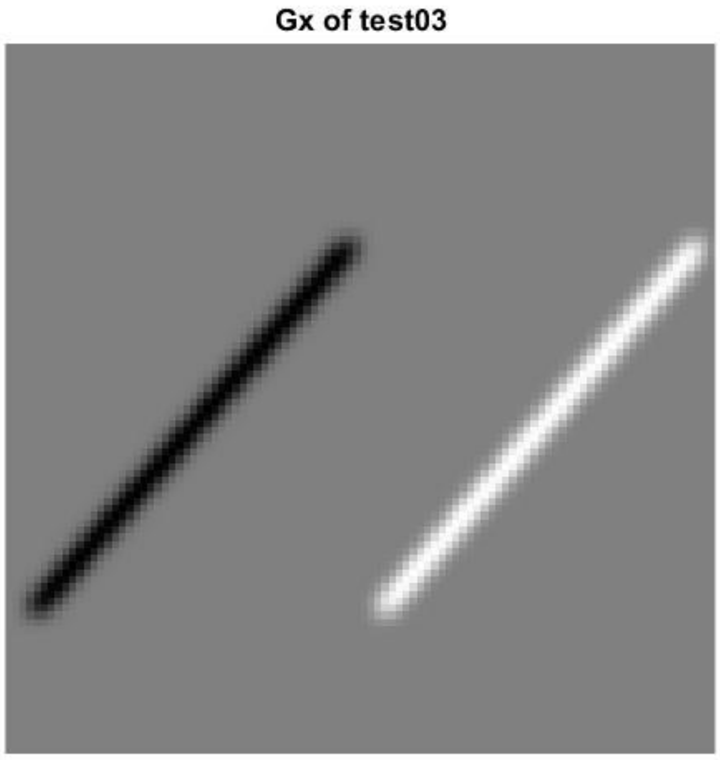
*Output of Gx image with test02 image:*



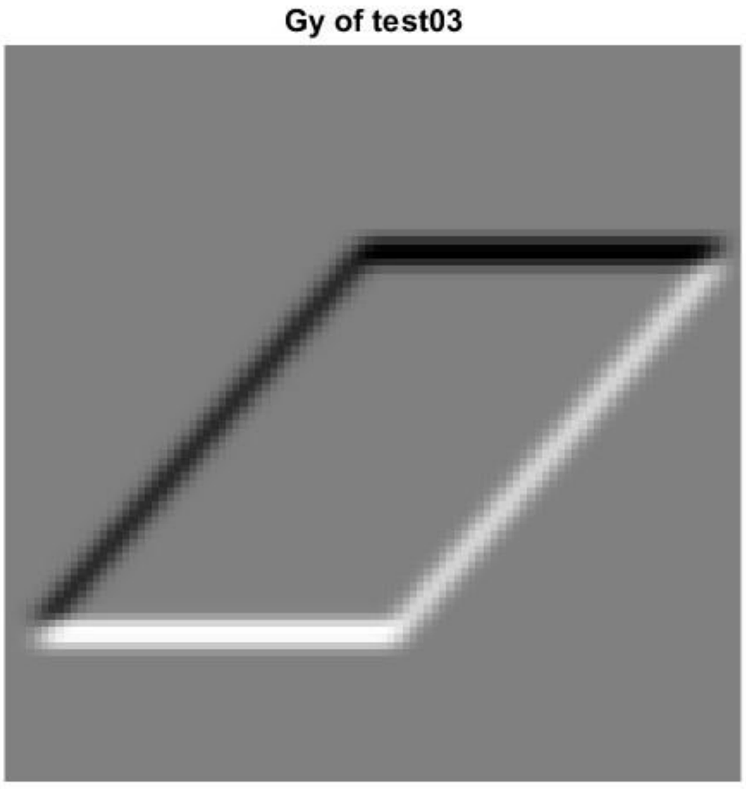
*Output of Gy image with test02 image:*



*Output of Gx image with test03 image:*

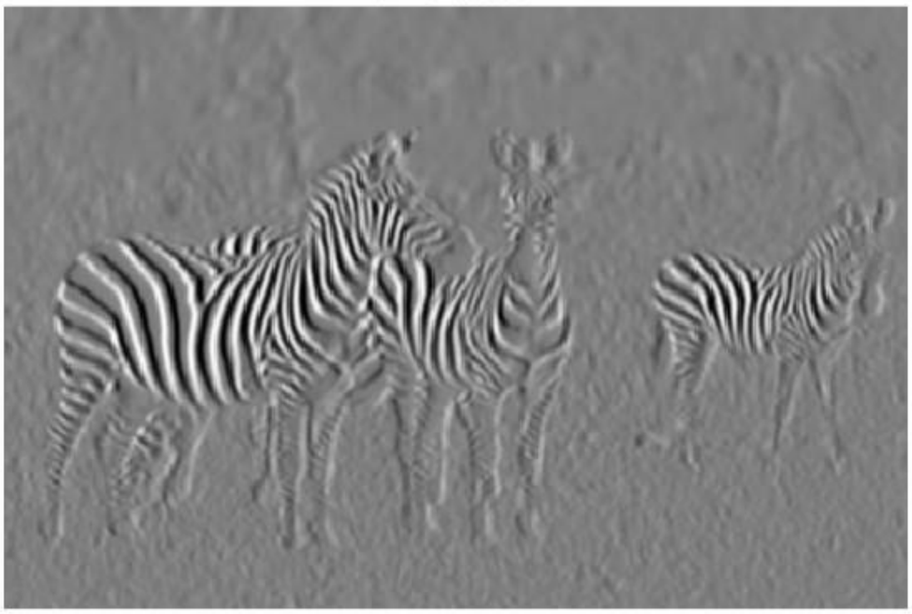


*Output of Gy image with test03 image:*



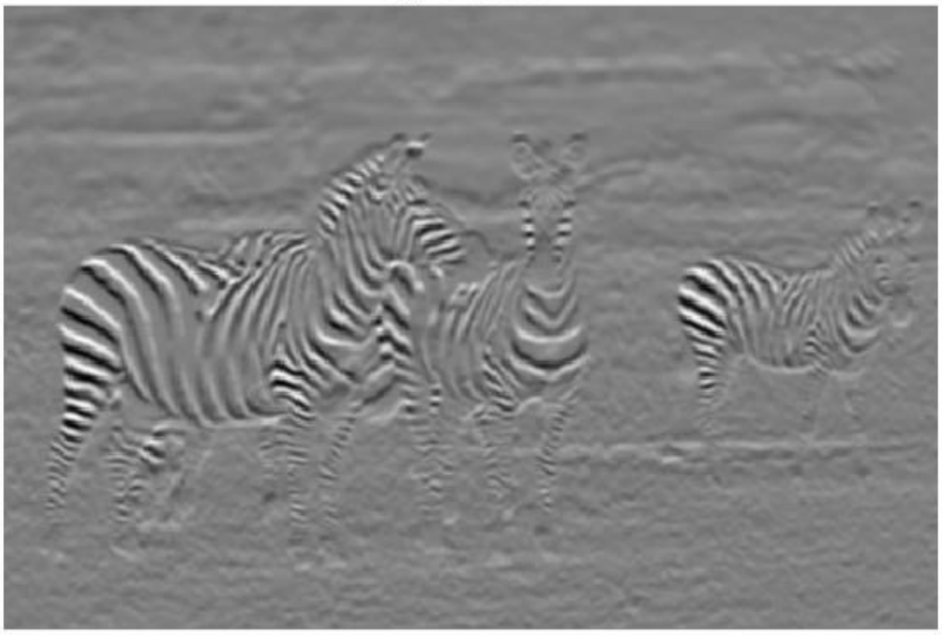
*Output of Gx image with test04 image:*

Gx of test04



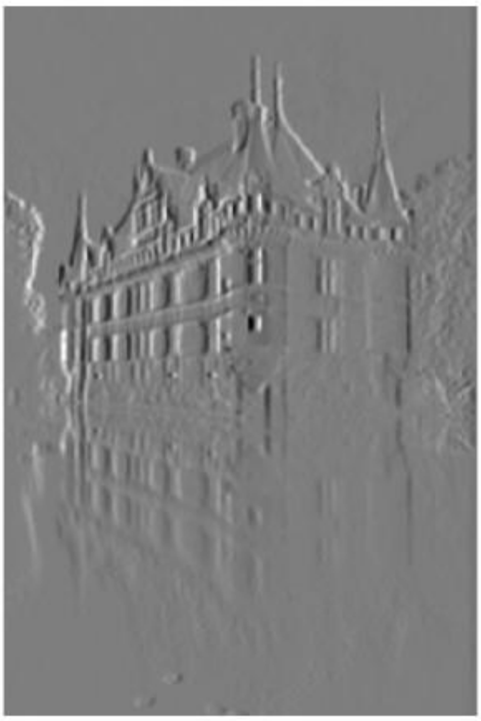
*Output of Gy image with test04 image:*

Gy of test04



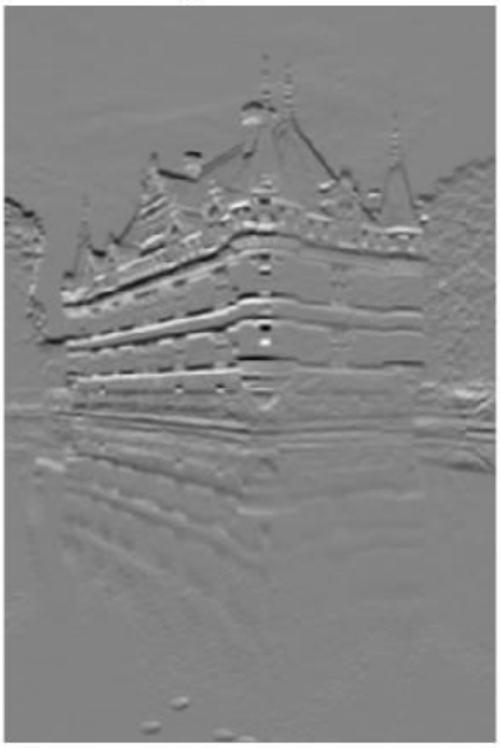
*Output of Gx image with test05 image:*

Gx of test05

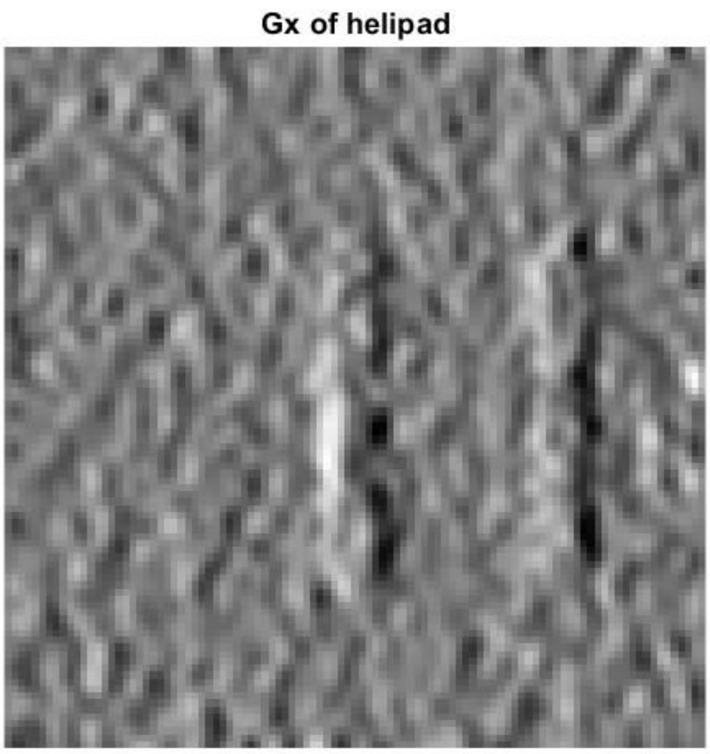


*Output of Gy image with test05 image:*

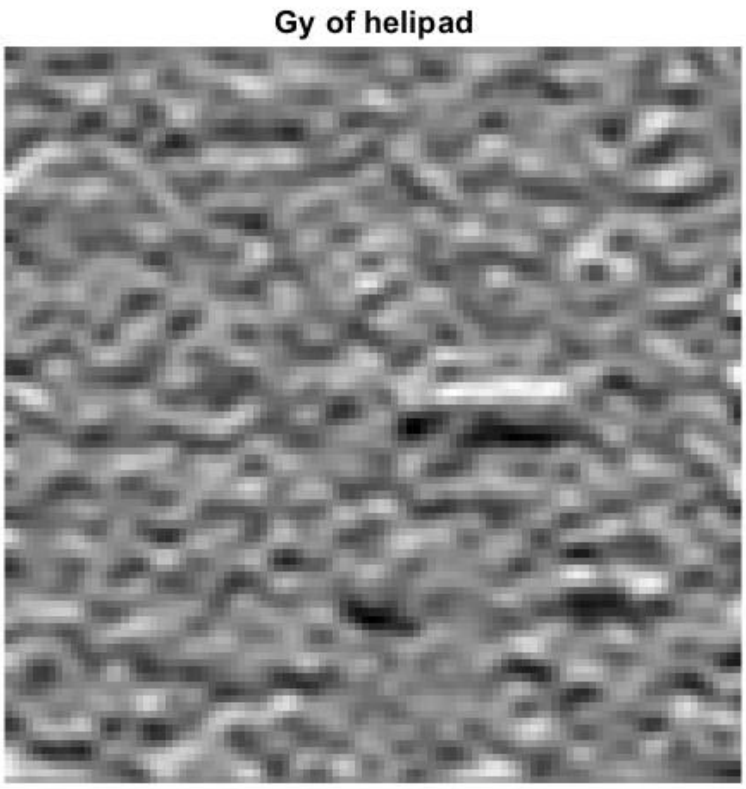
Gy of test05



*Output of Gx image with task06 image:*



*Output of Gy image with task06 image:*

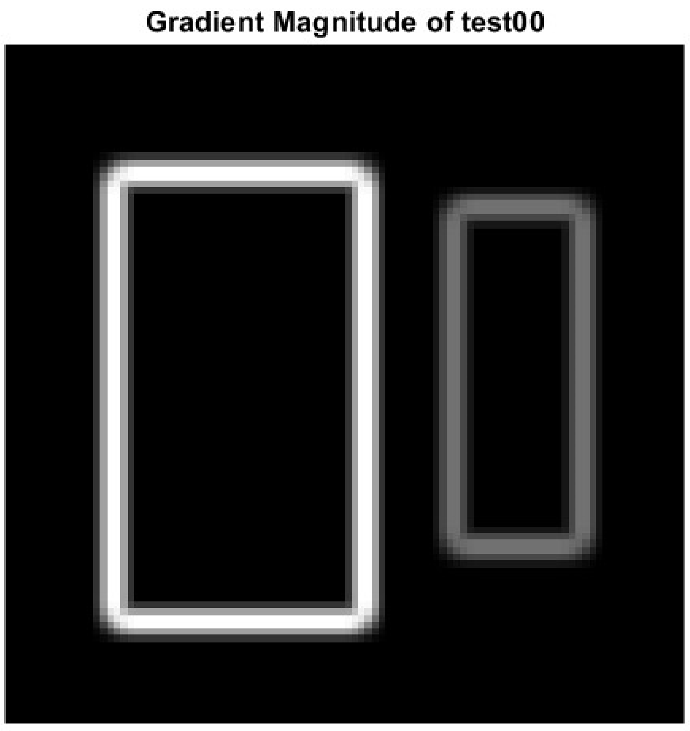


*Demonstrate your understanding by answering the question below:*
*Looking at the filter coefficients, explain how a sobel filter picks out horizontal edges?*

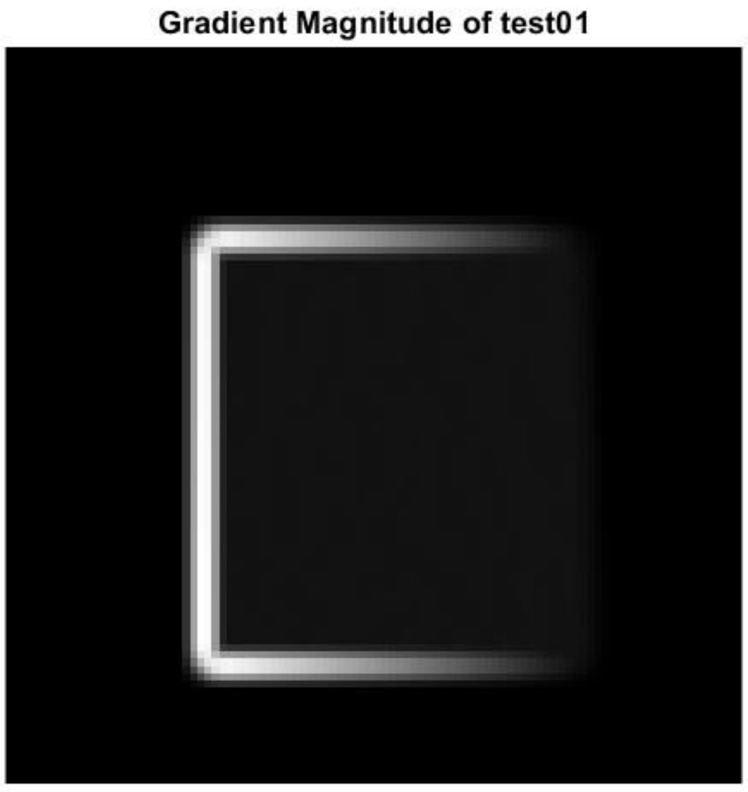
The Sobel filter picks out horizontal edges by computing the horizontal gradient of the image, which represents the rate of change of brightness in the horizontal direction.

By looking for areas where the horizontal gradient is high, the filter is able to identify horizontal edges in the image.

**Task 3 (1 mark):**
*Output for this task with test00 image:*

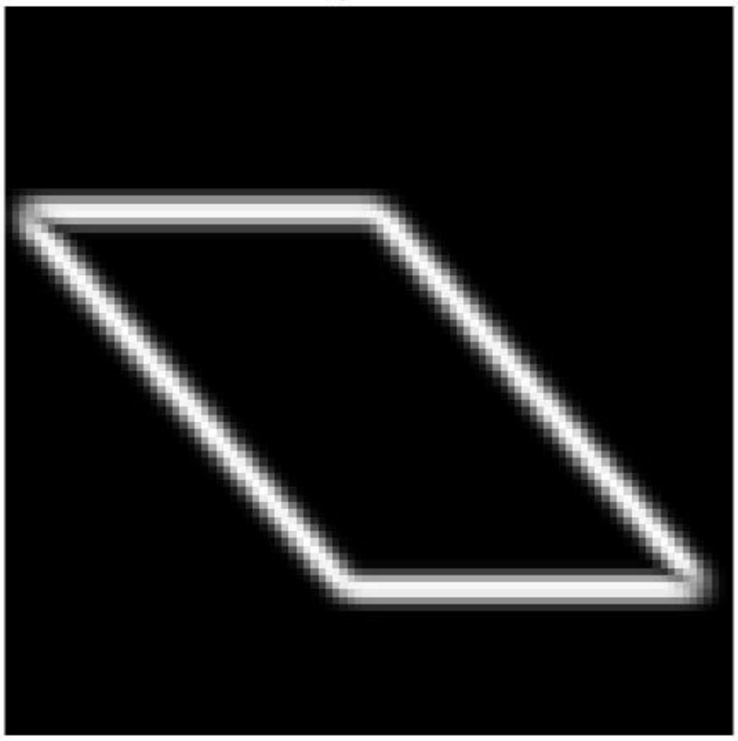


*Output for this task with test01 image:*



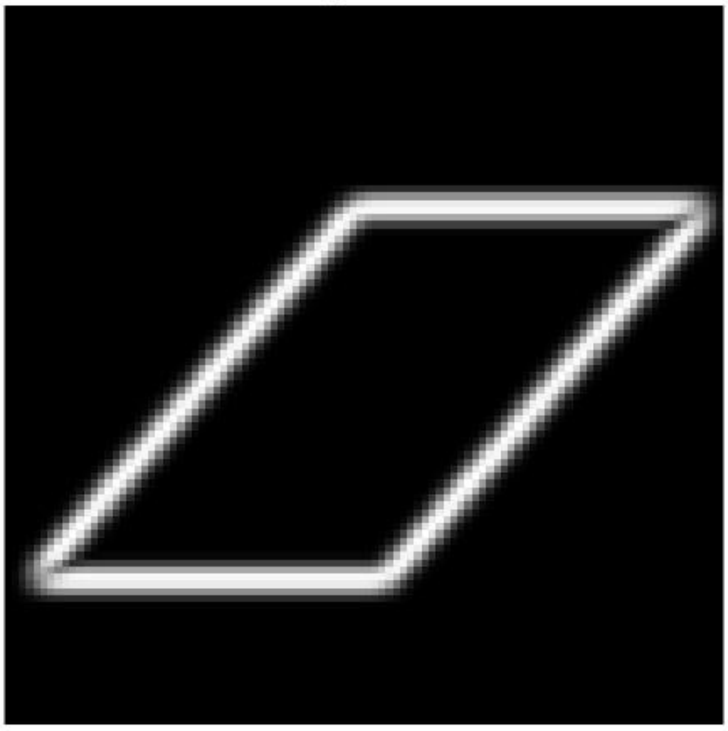
*Output for this task with test02 image:*

Gradient Magnitude of test02



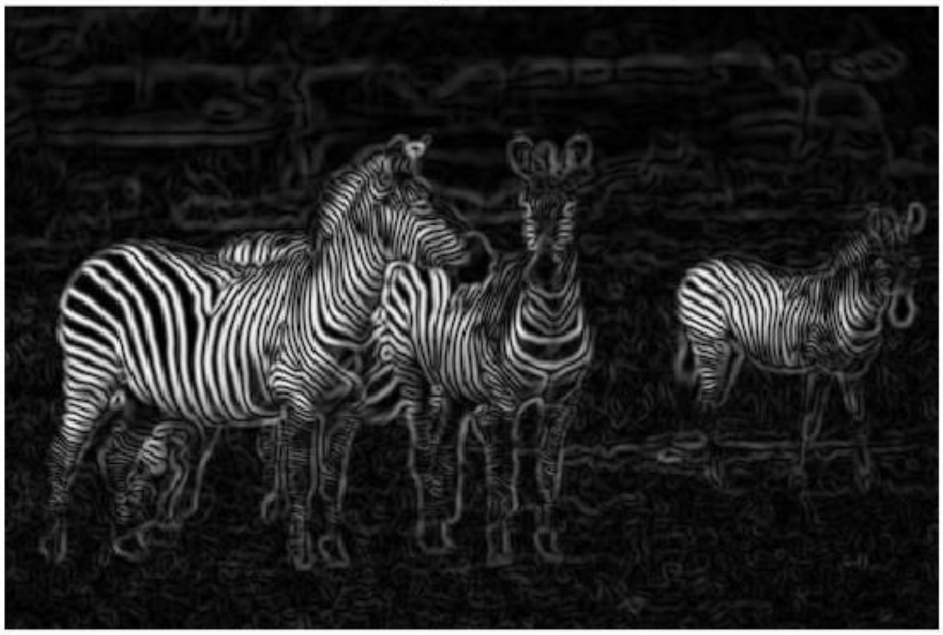
*Output for this task with test03 image:*

Gradient Magnitude of test03



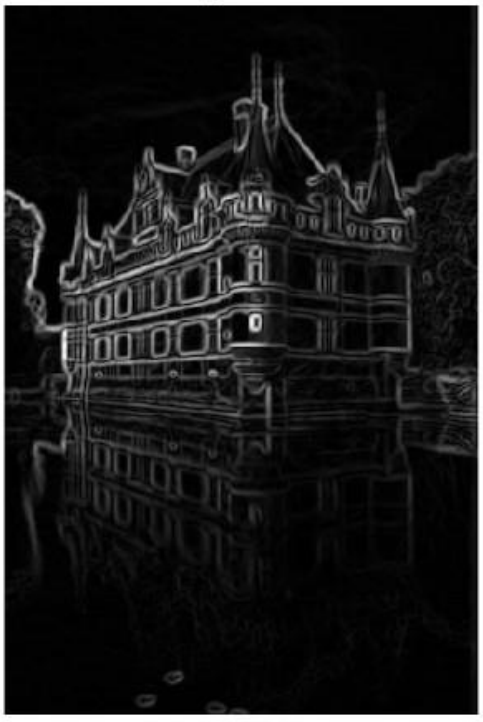
*Output for this task with test04 image:*

Gradient Magnitude of test04

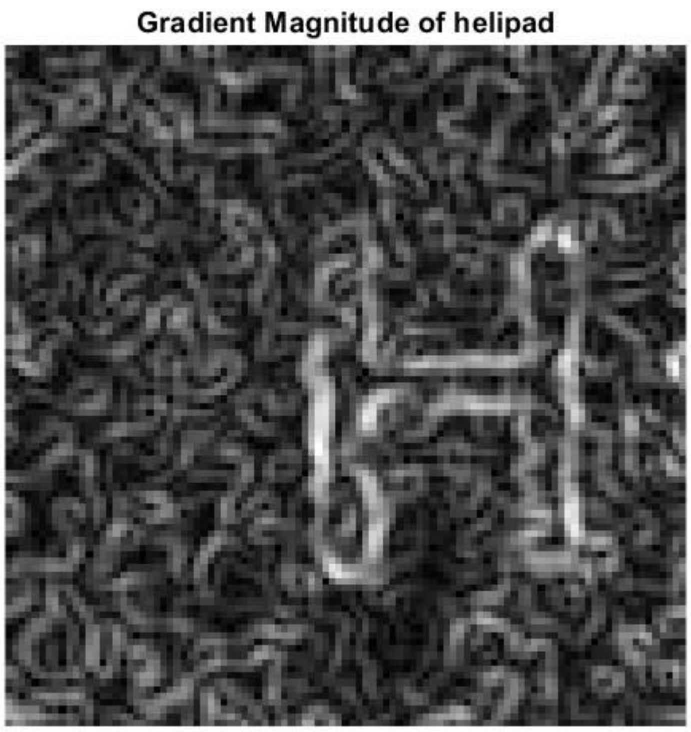


*Output for this task with test05 image:*

Gradient Magnitude of test05



*Output for this task with task06 image:*



*Demonstrate your understanding by answering the question below:*
*What differences in gradient magnitude would be observed for a corner, edge and solid region?*

The gradient magnitude is a measure of how quickly the intensity of an image changes in a given direction. For example, a high gradient magnitude indicates a rapid change in intensity, while a low gradient magnitude indicates a slow / no change in intensity.

In a corner, the intensity changes rapidly in both horizontal and vertical directions, resulting in a high gradient magnitude in both directions. The gradient magnitude is highest at the corner point and slowly decreases as it is moving away from the corner. Thus, a corner will show high gradient magnitudes in multiple directions.

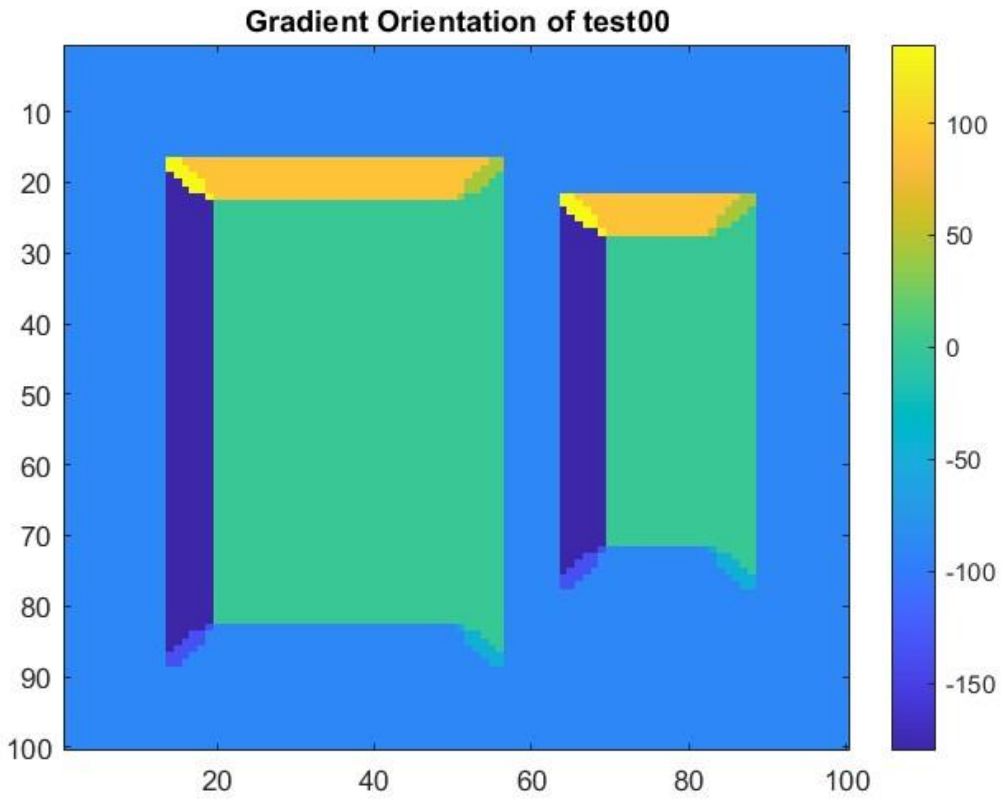
In an edge, the intensity changes rapidly in only one direction, resulting in a high gradient magnitude in that direction and a low gradient magnitude in the perpendicular direction. Thus, an edge will show high gradient magnitudes in only one direction.

In a solid region, there is no or very little change in intensity, resulting in a low gradient magnitude in all directions. Thus, a solid region will show low gradient magnitudes in all directions.

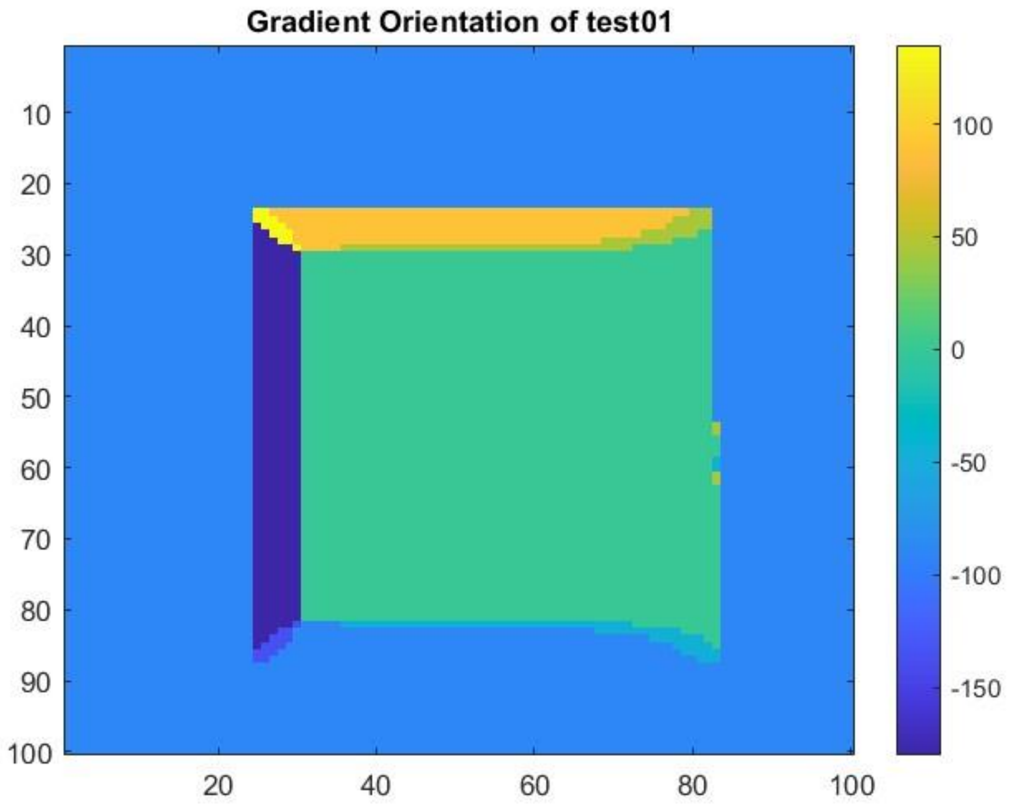
In a nutshell, a corner will show high gradient magnitudes in multiple directions, an edge will show high gradient magnitudes in only one direction, and a solid region will show low gradient magnitudes in all directions.

**Task 4 (1 mark):**

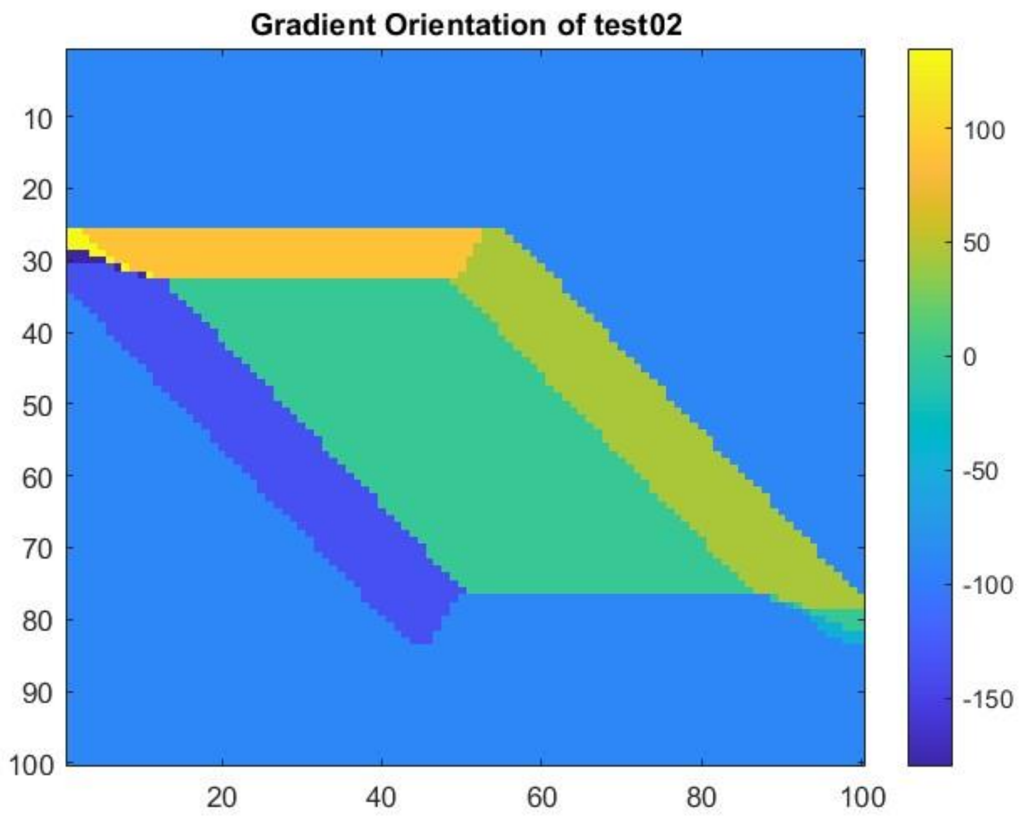
*Output for this task with test00 image:*



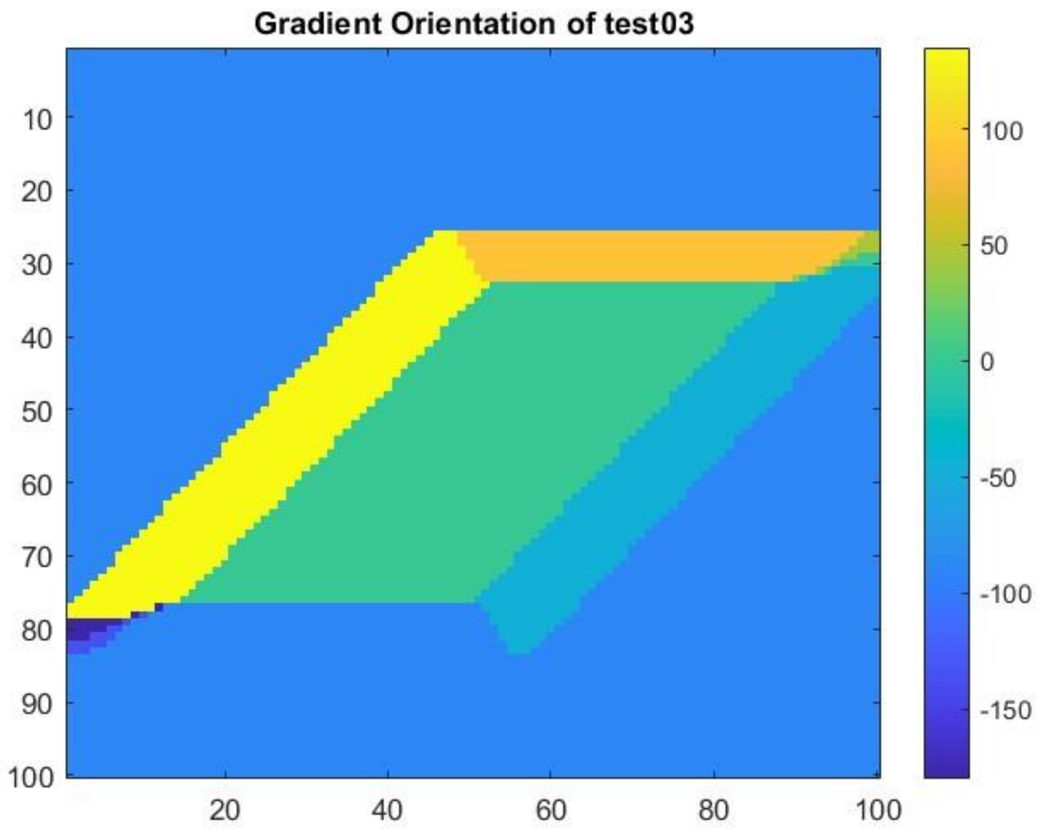
*Output for this task with test01 image:*



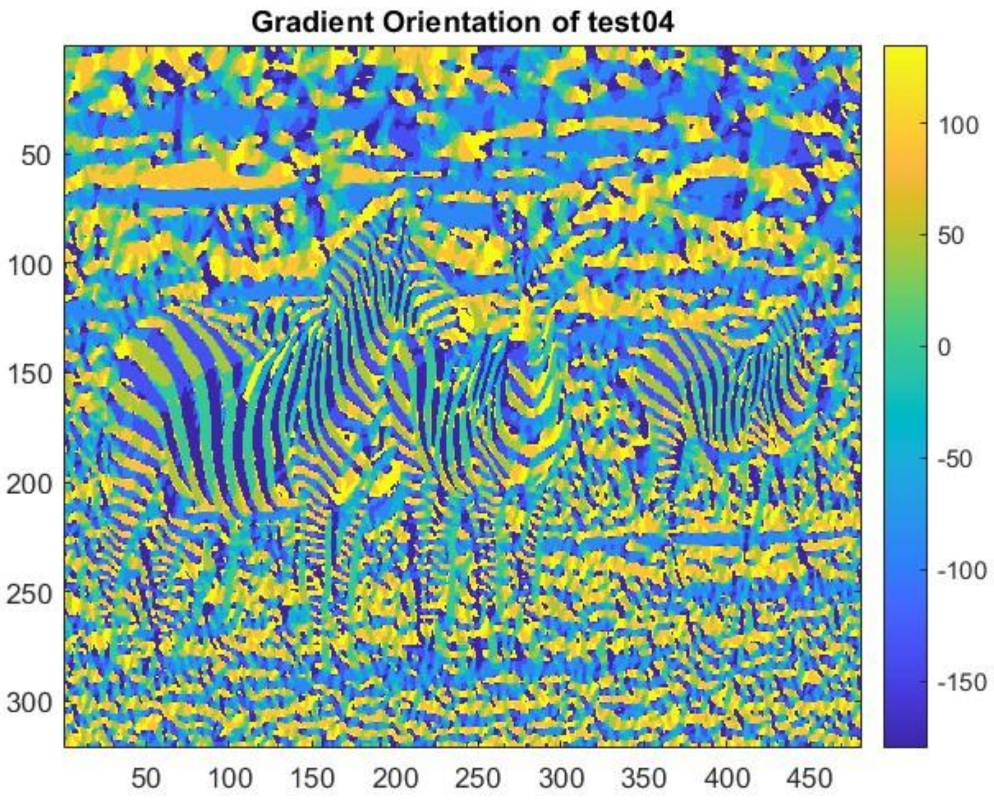
*Output for this task with test02 image:*



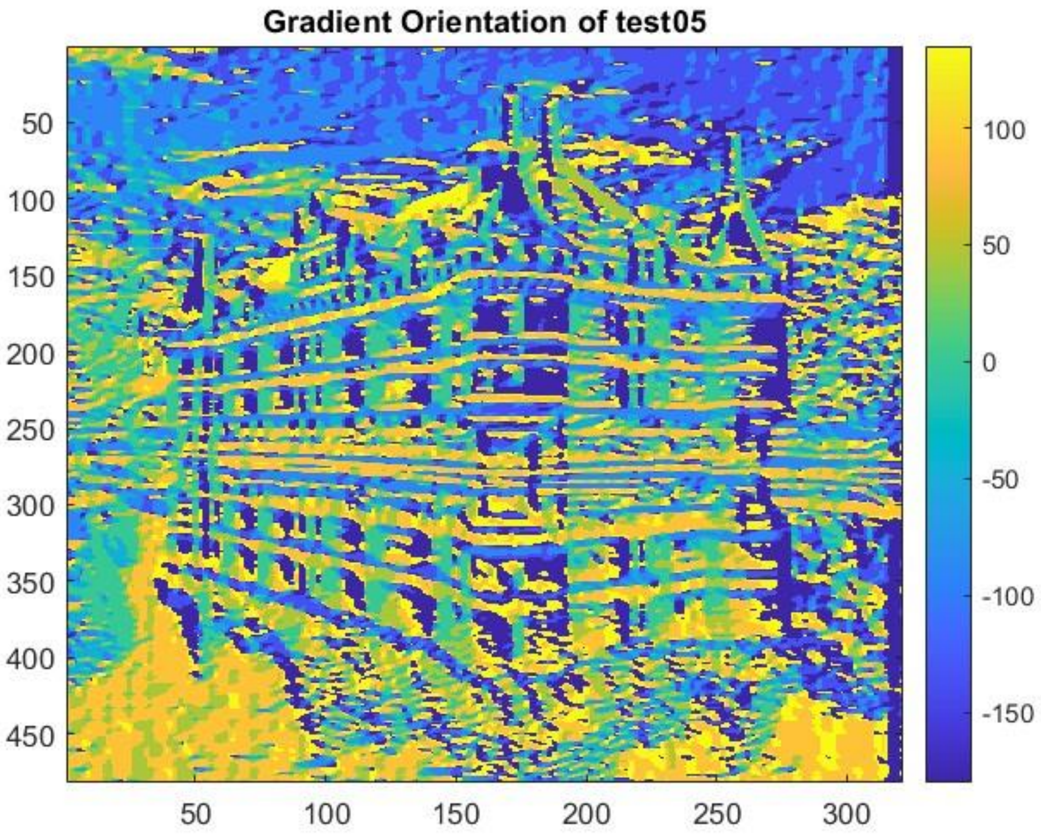
*Output for this task with test03 image:*



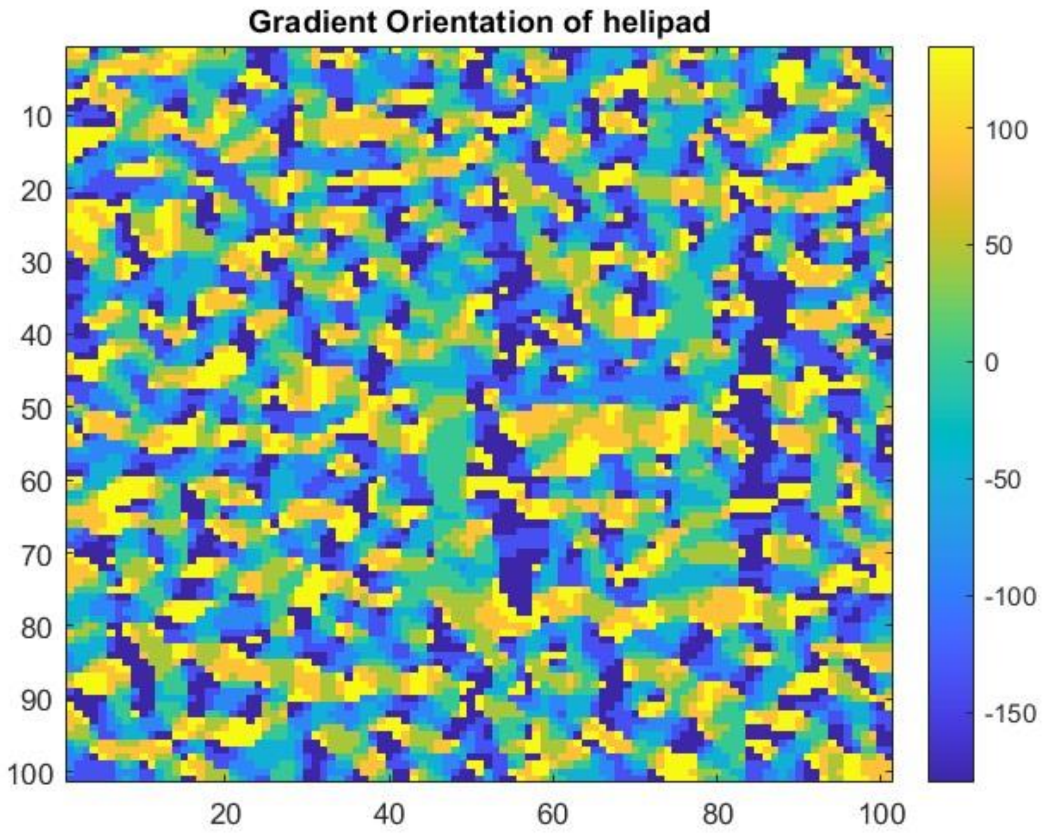
*Output for this task with test04 image:*



*Output for this task with test05 image:*



*Output for this task with task06 image:*

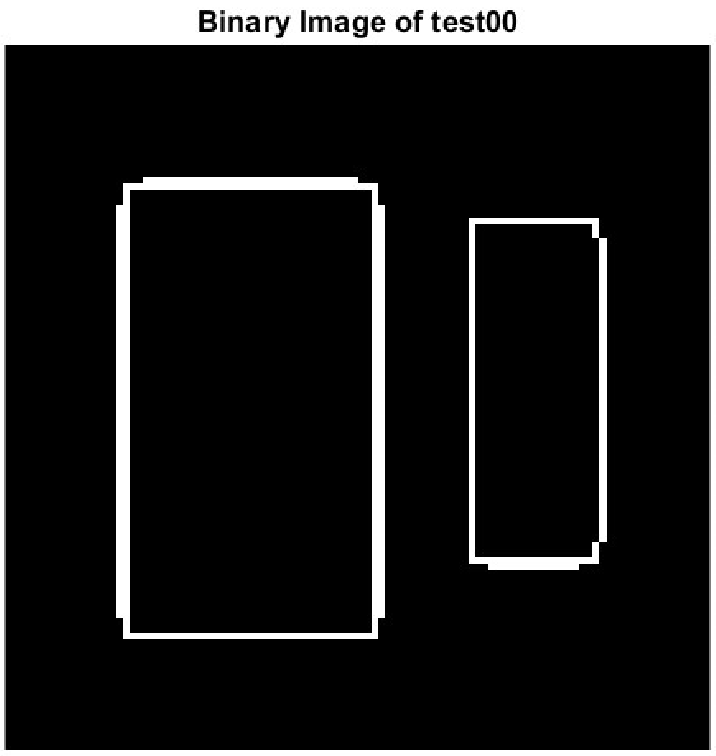


*Demonstrate your understanding by answering the question below:*
*How could the gradient orientation be used to obtain rotational invariance for patch matching?*

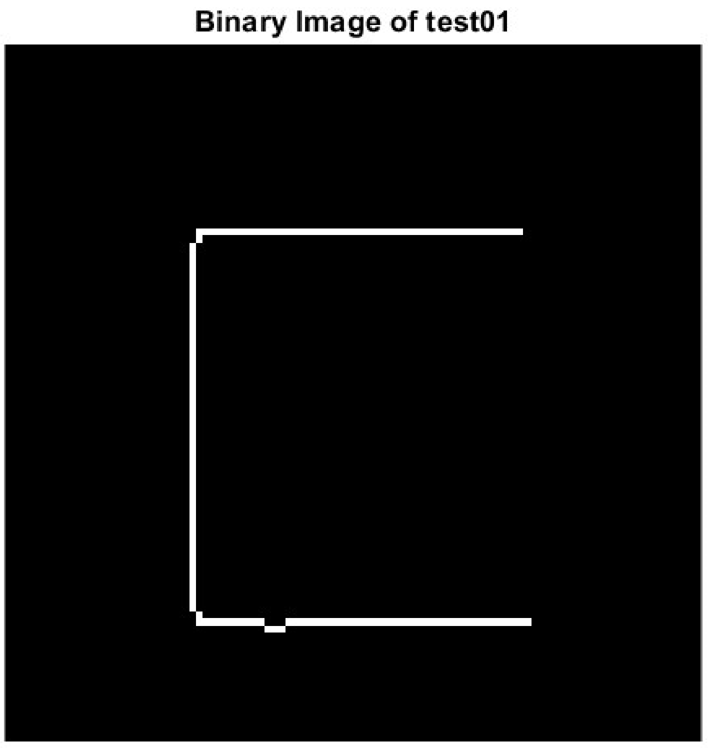
In patch matching, it is often desirable to achieve rotational invariance which means that the matching algorithm should be able to find correspondences between patches regardless of their orientation or rotation. One way to achieve this is by using the gradient orientation of the patches.

The gradient orientation of patches can be used to obtain rotation-invariant representations of patches for matching. By comparing the histograms of gradient orientations, we can achieve robust patch matching even when the patches are rotated or oriented differently.

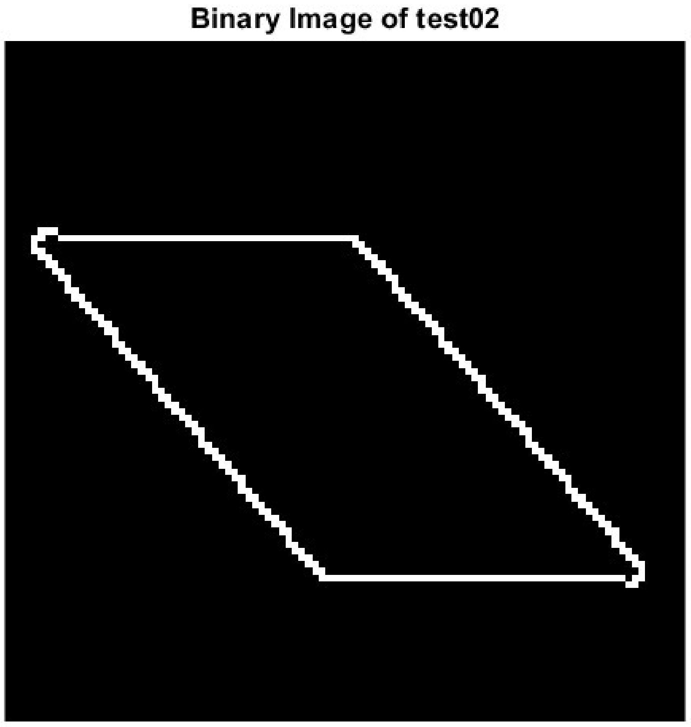
**Task 5 (2 marks):**
*Output for this task with test00 image:*



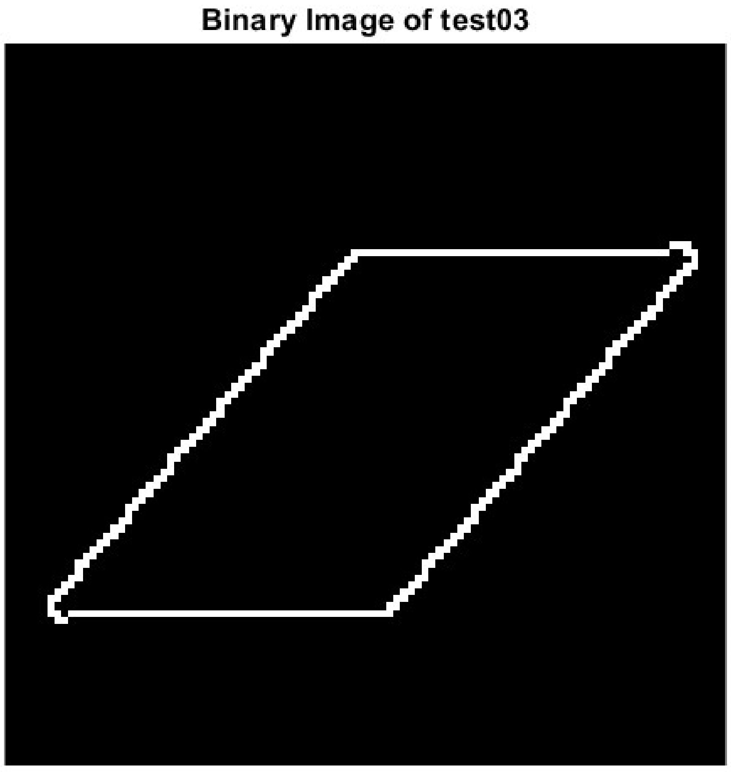
*Output for this task with test01 image:*



*Output for this task with test02 image:*

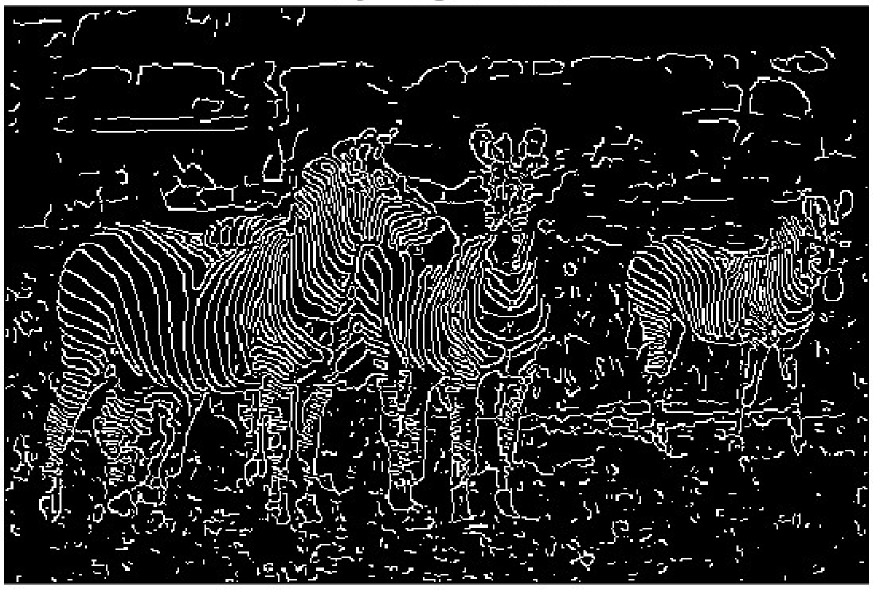


*Output for this task with test03 image:*



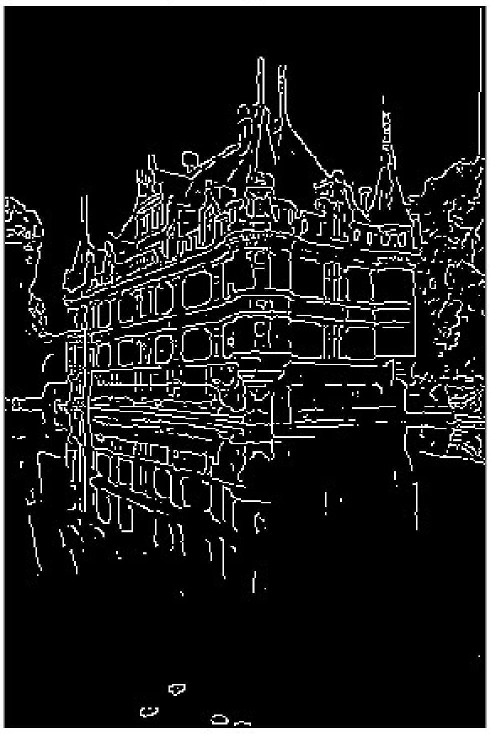
*Output for this task with test04 image:*

Binary Image of test04



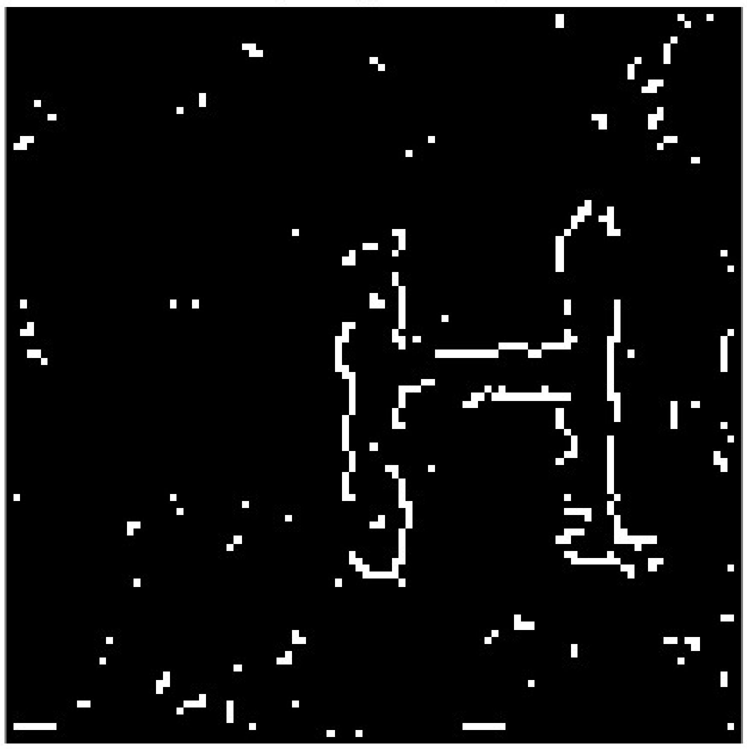
*Output for this task with test05 image:*

Binary Image of test05



*Output for this task with task06 image:*

Binary Image of helipad



*Demonstrate your understanding by answering the question below:*
*Explain how you chose the threshold for non-maximal suppression?*

The low threshold parameter sets the minimum threshold value for an edge to be considered part of the output. Edges with gradient magnitudes below this threshold are discarded.

The high threshold parameter sets the maximum threshold value for an edge to be considered part of the output, and is used to link edges together into continuous curves. If an edge has a gradient magnitude above the high threshold, it is considered a strong edge. If an edge has a gradient magnitude between the low and high thresholds, it is considered a weak edge.

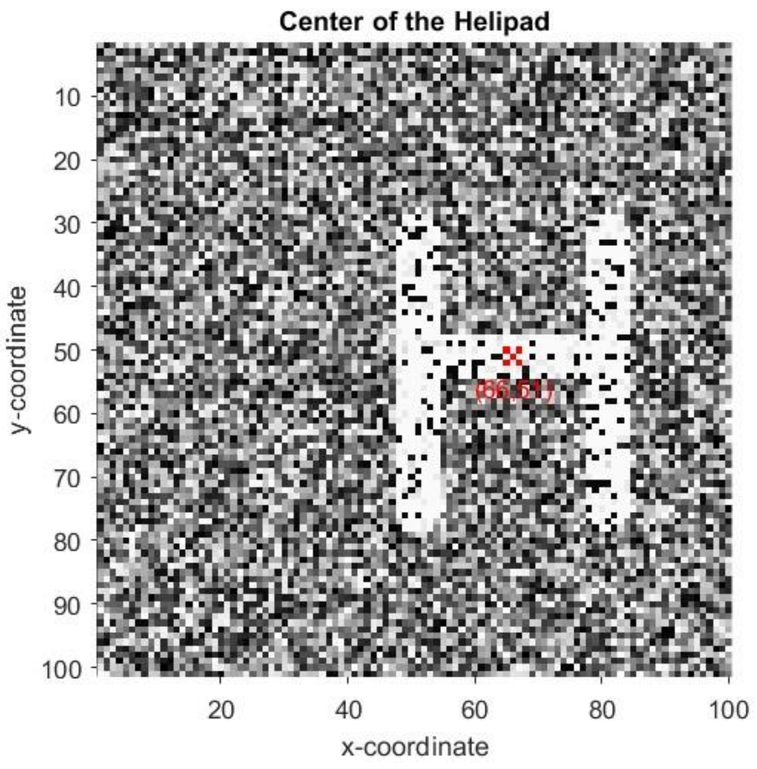
Basically, what I did was I tried to estimate through trial and error and see whether I can tweak the value based on the above two explanation.

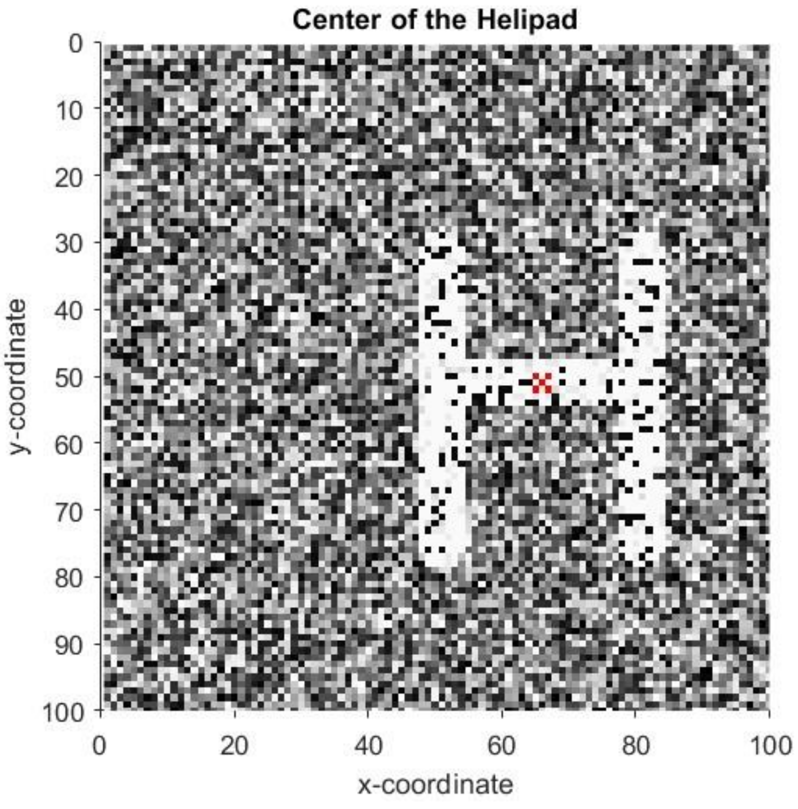
**Task 6 (1 mark):**

*Row where helipad centre is located (y-axis) = 51*

*Column where helipad centre is located (x-axis) = 66*

*Image with helipad centre plotted with an x:*





*Demonstrate your understanding by answering the question below:*
*How did you adapt your previous code to solve this problem?*

What I did was that I just used the code from the previous task, which includes calculating the gradient magnitude and the gradient orientation, find the suitable threshold to be used on my image and find the boundaries of the H symbol in my image. Later, I just plot an x on the supposed coordinates obtained via my algorithm.

**Function File Made:**

Paste your code in here.

```
% Written by Nigel Tan Jin Chun
% Last Modified: 15/3/2023
% Name of the file: myconv
% Purpose of the file: Implementing the function of the myconv
function
% Returns the convolution of the image with the kernel
% Used in Task 1

function B = myconv(Im, kernel)

    % Declaring and initialising all of the variables required to
loop through
    % the kernel and the image itself
    % Declaring and initialising the kernel first
    size_kernel = size(kernel);
    row_kernel = size_kernel(1);
    col_kernel = size_kernel(2);

    % Convert image to double percision
    Im_double = im2double(Im);

    % Ensuring that the output image will have the same size as the
input
    % image by padding the double precision values with zeros on all
sides
    % using the padarray function
    Im_p = padarray(Im_double, [floor(row_kernel/2),
floor(col_kernel/2)], 'replicate', 'both');

    % Getting the row and column of the image
    size_Im = size(Im_double);
    row_Im = size_Im(1);
    col_Im = size_Im(2);

    % Preallocating the variable for efficiency allocation purposes
    % B is basically the output image
    B = zeros(size_Im);

    % Using 4 loops to loop through the kernel and the image in
order to
    % perform the convolution operation
    % Looping through the pixels in the image
    for x = 1:row_Im
        for y = 1:col_Im

            % Looping through the pixels in the kernel
            for j = 1:row_kernel
                for k = 1:col_kernel

                    % For each pixel in the input image, the
corresponding pixel in the output image is computed by multiplying
the values in the kernel with the corresponding pixels in the padded
input image and summing the result.
```

```
                     B(x, y) = B(x, y) + (kernel(j,k) * Im_p(x + j -
1, y + k - 1));

                end
            end
        end
    end

end


% Written by Nigel Tan Jin Chun
% Last Modified: 15/3/2023
% Name of the file: myedge
% Purpose of the file: Implementing the function of the canny edge
% detector
% Returns a binary image with the edges
% Used in Task 5

function binary_Image = myedge(Im, G, theta, low_threshold,
high_threshold)

G_magnitude = padarray(G, [1, 1], 'replicate', 'both');
    G_direction = padarray(theta, [1, 1], 'replicate', 'both');

    [row, col] = size(G_magnitude);

    new_Im = zeros(size(G_magnitude));
    upper_pixel = 0;
    lower_pixel = 0;

    for i = 2:row-1
        for j = 2:col-1
            upper_pixel = 255;
            lower_pixel = 255;
            if ((G_direction(i, j) == 0) || (G_direction(i, j) ==
180) || (G_direction(i, j) == -180))
                upper_pixel = G_magnitude(i, j+1);
                lower_pixel = G_magnitude(i, j-1);

            elseif ((G_direction(i, j) == 45) || (G_direction(i, j)
== -135))
                upper_pixel = G_magnitude(i-1, j+1);
                lower_pixel = G_magnitude(i+1, j-1);

            elseif ((G_direction(i, j) == 90) || (G_direction(i, j)
== -90))
                upper_pixel = G_magnitude(i+1, j);
                lower_pixel = G_magnitude(i-1, j);

            elseif ((G_direction(i, j) == 135) || (G_direction(i, j)
== -45))
                upper_pixel = G_magnitude(i+1, j+1);
                lower_pixel = G_magnitude(i-1, j-1);
            end

            if ((G_magnitude(i,j) >= upper_pixel) &&
(G_magnitude(i,j) >= lower_pixel))
                new_Im(i,j) = G_magnitude(i,j);
            end
        end
    end

    high_T = max(max(new_Im)) * high_threshold;
    low_T = low_threshold * high_T;

    cannied_img = zeros(size(G_direction));
    idx_strong = new_Im >= high_T;
    idx_irrelevant = new_Im < low_T;
    idx_weak = (new_Im >= low_T) & (new_Im < high_T);

    cannied_img(idx_strong) = 255;
    cannied_img(idx_weak) = 128;
    cannied_img(idx_irrelevant) = 0;

    binary_Image = imbinarize(cannied_img, 127);
end
```

**Code for Task 1:**

```
Paste your code in here.
% Written by Nigel Tan Jin Chun
% Last Modified: 15/3/2023
% Name of the file: Lab1_Task1
% Purpose of the file: Implement Gaussian Blur
% Write a program that performs Gaussian blur on an input image
using the
% following 5x5 kernel, where B is the blurred version of input
image Im.

% Code should not use a pre-existing conv function

clear all;close all;clc;

% Defining the Gaussian kernel
matrix = [2,4,5,4,2;4,9,12,9,4;5,12,15,12,5;4,9,12,9,4;2,4,5,4,2];
const = 1/159;
kernel = const .* matrix;

%% Task 1 First Image
% Declaring and intialising the variable
% Using imread to read the given image
Im = imread('test00.png');

% convolving the two objects together
B = myconv(Im,kernel);

% Displaying the figure
figure(1);
imshow(B, 'InitialMagnification', 'fit');

% Labelling the figure
title("Test 00 Gaussian Blur Image");

%% Task 1 Second Image
% Declaring and intialising the variable
% Using imread to read the given image
Im = imread('test01.png');

% convolving the two objects together
B = myconv(Im,kernel);

% Displaying the figure
figure(2);
imshow(B , 'InitialMagnification', 'fit');

% Labelling the figure
title("Test 01 Gaussian Blur Image");

%% Task 1 Third Image
% Declaring and intialising the variable
% Using imread to read the given image
Im = imread('test02.png');
```

```
% Converting RGB to greyscale (If not you will have a red
background)
Im = rgb2gray(Im);

% convolving the two objects together
B = myconv(Im,kernel);

% Displaying the figure
figure(3);
imshow(B, 'InitialMagnification', 'fit');

% Labelling the figure
title("Test 02 Gaussian Blur Image");

%% Task 1 Fourth Image
% Declaring and intialising the variable
% Using imread to read the given image
Im = imread('test03.png');

% Converting RGB to greyscale (If not you will have a red
background)
Im = rgb2gray(Im);

% convolving the two objects together
B = myconv(Im,kernel);

% Displaying the figure
figure(4);
imshow(B, 'InitialMagnification', 'fit');

% Labelling the figure
title("Test 03 Gaussian Blur Image");

%% Task 1 Fifth Image
% Declaring and intialising the variable
% Using imread to read the given image
Im = imread('test04.jpg');

% convolving the two objects together
B = myconv(Im,kernel);

% Displaying the figure
figure(5);
imshow(B, 'InitialMagnification', 'fit');

% Labelling the figure
title("Test 04 Gaussian Blur Image");

%% Task 1 Sixth Image
% Declaring and intialising the variable
% Using imread to read the given image
Im = imread('test05.jpg');

% convolving the two objects together
B = myconv(Im,kernel);
```

```matlab
% Displaying the figure
figure(6);
imshow(B, 'InitialMagnification', 'fit');

% Labelling the figure
title("Test 05 Gaussian Blur Image");

%% Task 1 Seventh Image
% Declaring and intialising the variable
% Using imread to read the given image
Im = imread('task6_helipad.png');

% convolving the two objects together
B = myconv(Im,kernel);

% Displaying the figure
figure(7);
imshow(B, 'InitialMagnification', 'fit');

% Labelling the figure
title("Task 6 Gaussian Blur Image");
```

**Code for Task 2:**

Paste your code in here.

```
% Written by Nigel Tan Jin Chun
% Last Modified: 15/3/2023
% Name of the file: Lab1_Task2
% Purpose of the file: Calculate Image Gradient

clear all;close all;clc;

% Defining and initialising the variables
% Defining the Gaussian kernel
matrix = [2,4,5,4,2;4,9,12,9,4;5,12,15,12,5;4,9,12,9,4;2,4,5,4,2];
const = 1/159;
kernel = const .* matrix;

% Defining the Sobel Filter (Sobel Kernel Horizontal and Sobel
Kernal Vertical)
% Values are taken from L4 (Lecture 4)
x_sobel = [-1,0,1;-2,0,2;-1,0,1];
y_sobel = [-1,-2,-1;0,0,0;1,2,1];

%% First Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test00.png');

% Loading the image from the hard drive as a grayscale (single
channel
% image)
% Im = rgb2gray(Im);
Im = im2gray(Im);

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Checking the x and y gradient valeus to make sure that they are
correct
figure(1);

% [] argument scales the pixel values of the image to span the full
range of the colormap
% if not it will not properly display the image and it will not have
a
% uniform colour distribution
imshow(Gx, [], 'InitialMagnification', 'fit');
title("Gx of test00");
figure(2);
imshow(Gy, [], 'InitialMagnification', 'fit');
```

```
title("Gy of test00");

%% Second Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test01.png');

% Loading the image from the hard drive as a grayscale (single
channel
% image)
% Im = rgb2gray(Im);
Im = im2gray(Im);

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Checking the x and y gradient valeus to make sure that they are
correct
figure(3);

% [] argument scales the pixel values of the image to span the full
range of the colormap
% if not it will not properly display the image and it will not have
a
% uniform colour distribution
imshow(Gx, [], 'InitialMagnification', 'fit');
title("Gx of test01");
figure(4);
imshow(Gy, [], 'InitialMagnification', 'fit');
title("Gy of test01");

%% Third Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test02.png');

% Loading the image from the hard drive as a grayscale (single
channel
% image)
Im = rgb2gray(Im);
% Im = im2gray(Im);

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
```

```
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Checking the x and y gradient valeus to make sure that they are
correct
figure(5);

% [] argument scales the pixel values of the image to span the full
range of the colormap
% if not it will not properly display the image and it will not have
a
% uniform colour distribution
imshow(Gx, [], 'InitialMagnification', 'fit');
title("Gx of test02");
figure(6);
imshow(Gy, [], 'InitialMagnification', 'fit');
title("Gy of test02");

%% Fourth Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test03.png');

% Loading the image from the hard drive as a grayscale (single
channel
% image)
Im = rgb2gray(Im);
% Im = im2gray(Im);

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Checking the x and y gradient valeus to make sure that they are
correct
figure(7);

% [] argument scales the pixel values of the image to span the full
range of the colormap
% if not it will not properly display the image and it will not have
a
% uniform colour distribution
imshow(Gx, [], 'InitialMagnification', 'fit');
title("Gx of test03");
figure(8);
imshow(Gy, [], 'InitialMagnification', 'fit');
title("Gy of test03");
```

```
%% Fifth Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test04.jpg');

% Loading the image from the hard drive as a grayscale (single
channel
% image)
% Im = rgb2gray(Im);
Im = im2gray(Im);

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Checking the x and y gradient valeus to make sure that they are
correct
figure(9);

% [] argument scales the pixel values of the image to span the full
range of the colormap
% if not it will not properly display the image and it will not have
a
% uniform colour distribution
imshow(Gx, [], 'InitialMagnification', 'fit');
title("Gx of test04");
figure(10);
imshow(Gy, [], 'InitialMagnification', 'fit');
title("Gy of test04");

%% Sixth Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test05.jpg');

% Loading the image from the hard drive as a grayscale (single
channel
% image)
% Im = rgb2gray(Im);
Im = im2gray(Im);

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
```

```
Gy = myconv(B,y_sobel);

% Checking the x and y gradient valeus to make sure that they are
correct
figure(11);

% [] argument scales the pixel values of the image to span the full
range of the colormap
% if not it will not properly display the image and it will not have
a
% uniform colour distribution
imshow(Gx, [], 'InitialMagnification', 'fit');
title("Gx of test05");
figure(12);
imshow(Gy, [], 'InitialMagnification', 'fit');
title("Gy of test05");

%% Seventh Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('task6_helipad.png');

% Loading the image from the hard drive as a grayscale (single
channel
% image)
% Im = rgb2gray(Im);
Im = im2gray(Im);

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Checking the x and y gradient valeus to make sure that they are
correct
figure(13);

% [] argument scales the pixel values of the image to span the full
range of the colormap
% if not it will not properly display the image and it will not have
a
% uniform colour distribution
imshow(Gx, [], 'InitialMagnification', 'fit');
title("Gx of helipad");
figure(14);
imshow(Gy, [], 'InitialMagnification', 'fit');
title("Gy of helipad");
```

**Code for Task 3:**

Paste your code in here.

```
% Written by Nigel Tan Jin Chun
% Last Modified: 15/3/2023
% Name of the file: Lab1_Task3
% Purpose of the file: Calculate gradient magnitude

clear all;close all;clc;

% Defining the Gaussian kernel
matrix = [2,4,5,4,2;4,9,12,9,4;5,12,15,12,5;4,9,12,9,4;2,4,5,4,2];
const = 1/159;
kernel = const .* matrix;

% Defining the Sobel Filter (Sobel Kernel Horizontal and Sobel
Kernal Vertical)
% Values are taken from L4 (Lecture 4)
x_sobel = [-1,0,1;-2,0,2;-1,0,1];
y_sobel = [-1,-2,-1;0,0,0;1,2,1];

%% First Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test00.png');

% Loading the image from the hard drive as a grayscale (single
channel
% image)
% Im = rgb2gray(Im);
Im = im2gray(Im);

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Calculate the gradient magnitude
G = sqrt(Gx.^2 + Gy.^2);

% Showing the gradient magnitude on screen as a greyscale image
figure(1);
imshow(G,[], 'InitialMagnification', 'fit');
title("Gradient Magnitude of test00");

%% Second Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test01.png');
```

```
% Loading the image from the hard drive as a grayscale (single
channel
% image)
% Im = rgb2gray(Im);
Im = im2gray(Im);

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Calculate the gradient magnitude
G = sqrt(Gx.^2 + Gy.^2);

% Showing the gradient magnitude on screen as a greyscale image
figure(2);
imshow(G,[], 'InitialMagnification', 'fit');
title("Gradient Magnitude of test01");

%% Third Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test02.png');

% Loading the image from the hard drive as a grayscale (single
channel
% image)
Im = rgb2gray(Im);
% Im = im2gray(Im);

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Calculate the gradient magnitude
G = sqrt(Gx.^2 + Gy.^2);

% Showing the gradient magnitude on screen as a greyscale image
figure(3);
imshow(G,[], 'InitialMagnification', 'fit');
title("Gradient Magnitude of test02");

%% Fourth Image
% Declaring and intialising the variable
```

```matlab
% Using imread to read the given image and store it in a variable
Im = imread('test03.png');

% Loading the image from the hard drive as a grayscale (single
channel
% image)
Im = rgb2gray(Im);
% Im = im2gray(Im);

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Calculate the gradient magnitude
G = sqrt(Gx.^2 + Gy.^2);

% Showing the gradient magnitude on screen as a greyscale image
figure(4);
imshow(G,[], 'InitialMagnification', 'fit');
title("Gradient Magnitude of test03");

%% Fifth Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test04.jpg');

% Loading the image from the hard drive as a grayscale (single
channel
% image)
% Im = rgb2gray(Im);
Im = im2gray(Im);

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Calculate the gradient magnitude
G = sqrt(Gx.^2 + Gy.^2);

% Showing the gradient magnitude on screen as a greyscale image
figure(5);
imshow(G,[], 'InitialMagnification', 'fit');
title("Gradient Magnitude of test04");
```

```
%% Sixth Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test05.jpg');

% Loading the image from the hard drive as a grayscale (single
channel
% image)
% Im = rgb2gray(Im);
Im = im2gray(Im);

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Calculate the gradient magnitude
G = sqrt(Gx.^2 + Gy.^2);

% Showing the gradient magnitude on screen as a greyscale image
figure(6);
imshow(G,[], 'InitialMagnification', 'fit');
title("Gradient Magnitude of test05");

%% Seventh Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('task6_helipad.png');

% Loading the image from the hard drive as a grayscale (single
channel
% image)
% Im = rgb2gray(Im);
Im = im2gray(Im);

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Calculate the gradient magnitude
G = sqrt(Gx.^2 + Gy.^2);

% Showing the gradient magnitude on screen as a greyscale image
```

```
figure(7);
imshow(G,[], 'InitialMagnification', 'fit');
title("Gradient Magnitude of helipad");
```

**Code for Task 4:**

Paste your code in here.

```
% Written by Nigel Tan Jin Chun
% Last Modified: 15/3/2023
% Name of the file: Lab1_Task4
% Purpose of the file: Calculate gradient orientation

clear all;close all;clc;

% Defining the Gaussian kernel
matrix = [2,4,5,4,2;4,9,12,9,4;5,12,15,12,5;4,9,12,9,4;2,4,5,4,2];
const = 1/159;
kernel = const .* matrix;

% Defining the Sobel Filter (Sobel Kernel Horizontal and Sobel
Kernal Vertical)
% Values are taken from L4 (Lecture 4)
x_sobel = [-1,0,1;-2,0,2;-1,0,1];
y_sobel = [-1,-2,-1;0,0,0;1,2,1];

%% First Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test00.png');

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Calculating the gradient orientation
theta = atan2d(-Gy, Gx);

% % Gradient orientations should be rounded to the nearest 45
degrees
theta = round(theta./45)*45;
theta(theta == 180) = -180;

% Showing the gradient magnitude on screen as a greyscale image
figure(1);
imagesc(theta);
% colourmap(theta);
colorbar;
title("Gradient Orientation of test00");

%% Second Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test01.png');
```

```
% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Calculating the gradient orientation
theta = atan2d(-Gy, Gx);

% % Gradient orientations should be rounded to the nearest 45
degrees
theta = round(theta./45)*45;
theta(theta == 180) = -180;

% Showing the gradient magnitude on screen as a greyscale image
figure(2);
imagesc(theta);
% colourmap(theta);
colorbar;
title("Gradient Orientation of test01");

%% Third Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test02.png');

% Loading the image from the hard drive as a grayscale (single
channel
% image)
Im = rgb2gray(Im);

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Calculating the gradient orientation
theta = atan2d(-Gy, Gx);

% % Gradient orientations should be rounded to the nearest 45
degrees
theta = round(theta./45)*45;
theta(theta == 180) = -180;

% Showing the gradient magnitude on screen as a greyscale image
figure(3);
```

```
imagesc(theta);
% colourmap(theta);
colorbar;
title("Gradient Orientation of test02");

%% Fourth Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test03.png');

% Loading the image from the hard drive as a grayscale (single
channel
% image)
Im = rgb2gray(Im);

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Calculating the gradient orientation
theta = atan2d(-Gy, Gx);

% % Gradient orientations should be rounded to the nearest 45
degrees
theta = round(theta./45)*45;
theta(theta == 180) = -180;

% Showing the gradient magnitude on screen as a greyscale image
figure(4);
imagesc(theta);
% colourmap(theta);
colorbar;
title("Gradient Orientation of test03");

%% Fifth Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test04.jpg');

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);
```

```matlab
% Calculating the gradient orientation
theta = atan2d(-Gy, Gx);

% % Gradient orientations should be rounded to the nearest 45
degrees
theta = round(theta./45)*45;
theta(theta == 180) = -180;

% Showing the gradient magnitude on screen as a greyscale image
figure(5);
imagesc(theta);
% colourmap(theta);
colorbar;
title("Gradient Orientation of test04");

%% Sixth Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test05.jpg');

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Calculate the gradient magnitude
G = sqrt(Gx.^2 + Gy.^2);

% Calculating the gradient orientation
theta = atan2d(-Gy, Gx);

% % Gradient orientations should be rounded to the nearest 45
degrees
theta = round(theta./45)*45;
theta(theta == 180) = -180;

% Showing the gradient magnitude on screen as a greyscale image
figure(6);
imagesc(theta);
% colourmap(theta);
colorbar;
title("Gradient Orientation of test05");

%% Seventh Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('task6_helipad.png');

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
```

```
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Calculating the gradient orientation
theta = atan2d(-Gy, Gx);

% % Gradient orientations should be rounded to the nearest 45
degrees
theta = round(theta./45)*45;
theta(theta == 180) = -180;

% Showing the gradient magnitude on screen as a greyscale image
figure(7);
imagesc(theta);
% colourmap(theta);
colorbar;
title("Gradient Orientation of helipad");
```

**Code for Task 5:**

```
Paste your code in here.
% Written by Nigel Tan Jin Chun
% Last Modified: 15/3/2023
% Name of the file: Lab1_Task5
% Purpose of the file: Perform non-maxima suppresion and handling

clear all;close all;clc;

% Defining the Gaussian kernel
matrix = [2,4,5,4,2;4,9,12,9,4;5,12,15,12,5;4,9,12,9,4;2,4,5,4,2];
const = 1/159;
kernel = const .* matrix;

% Defining the Sobel Filter (Sobel Kernel Horizontal and Sobel
Kernal Vertical)
% Values are taken from L4 (Lecture 4)
x_sobel = [-1,0,1;-2,0,2;-1,0,1];
y_sobel = [-1,-2,-1;0,0,0;1,2,1];

% Need to zero any pixel that is not greater in terms of gradient
magnitude
% than both pixels on either side of its gradient orientation
% https://en.wikipedia.org/wiki/Canny_edge_detector
%% First Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test00.png');

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Calculate the gradient magnitude
G = sqrt(Gx.^2 + Gy.^2);

% Calculating the gradient orientation
theta = atan2d(-Gy, Gx);

% Gradient orientations should be rounded to the nearest 45 degrees
theta = round(theta./45)*45;
theta(theta == 180) = -180;

% Performing non maxima suppression
low_threshold = 0.12;
high_threshold = 0.8;
binary_Image = myedge(Im, G, theta, low_threshold, high_threshold);

% Showing the gradient magnitude on screen as a greyscale image
```

```
figure(1);
imshow(binary_Image, 'InitialMagnification', 'fit');
title("Binary Image of test00");

%% Second Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test01.png');

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Calculate the gradient magnitude
G = sqrt(Gx.^2 + Gy.^2);

% Calculating the gradient orientation
theta = atan2d(-Gy, Gx);

% Gradient orientations should be rounded to the nearest 45 degrees
theta = round(theta./45)*45;
theta(theta == 180) = -180;

% Performing non maxima suppression
low_threshold = 0.12;
high_threshold = 0.8;
binary_Image = myedge(Im, G, theta, low_threshold, high_threshold);

% Showing the gradient magnitude on screen as a greyscale image
figure(2);
imshow(binary_Image, 'InitialMagnification', 'fit');
title("Binary Image of test01");

%% Third Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test02.png');

% Loading the image from the hard drive as a grayscale (single
channel
% image)
Im = rgb2gray(Im);

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
```

```
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Calculate the gradient magnitude
G = sqrt(Gx.^2 + Gy.^2);

% Calculating the gradient orientation
theta = atan2d(-Gy, Gx);

% Gradient orientations should be rounded to the nearest 45 degrees
theta = round(theta./45)*45;
theta(theta == 180) = -180;

% Performing non maxima suppression
low_threshold = 0.12;
high_threshold = 0.8;
binary_Image = myedge(Im, G, theta, low_threshold, high_threshold);

% Showing the gradient magnitude on screen as a greyscale image
figure(3);
imshow(binary_Image, 'InitialMagnification', 'fit');
title("Binary Image of test02");

%% Fourth Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test03.png');

% Loading the image from the hard drive as a grayscale (single
channel
% image)
Im = rgb2gray(Im);

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Calculate the gradient magnitude
G = sqrt(Gx.^2 + Gy.^2);

% Calculating the gradient orientation
theta = atan2d(-Gy, Gx);

% Gradient orientations should be rounded to the nearest 45 degrees
theta = round(theta./45)*45;
theta(theta == 180) = -180;

% Performing non maxima suppression
```

```matlab
low_threshold = 0.12;
high_threshold = 0.8;
binary_Image = myedge(Im, G, theta, low_threshold, high_threshold);

% Showing the gradient magnitude on screen as a greyscale image
figure(4);
imshow(binary_Image, 'InitialMagnification', 'fit');
title("Binary Image of test03");

%% Fifth Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test04.jpg');

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Calculate the gradient magnitude
G = sqrt(Gx.^2 + Gy.^2);

% Calculating the gradient orientation
theta = atan2d(-Gy, Gx);

% Gradient orientations should be rounded to the nearest 45 degrees
theta = round(theta./45)*45;
theta(theta == 180) = -180;

% Performing non maxima suppression
low_threshold = 0.12;
high_threshold = 0.8;
binary_Image = myedge(Im, G, theta, low_threshold, high_threshold);

% Showing the gradient magnitude on screen as a greyscale image
figure(5);
imshow(binary_Image, 'InitialMagnification', 'fit');
title("Binary Image of test04");

%% Sixth Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('test05.jpg');

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
```

```
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Calculate the gradient magnitude
G = sqrt(Gx.^2 + Gy.^2);

% Calculating the gradient orientation
theta = atan2d(-Gy, Gx);

% Gradient orientations should be rounded to the nearest 45 degrees
theta = round(theta./45)*45;
theta(theta == 180) = -180;

% Performing non maxima suppression
low_threshold = 0.12;
high_threshold = 0.8;
binary_Image = myedge(Im, G, theta, low_threshold, high_threshold);

% Showing the gradient magnitude on screen as a greyscale image
figure(6);
imshow(binary_Image, 'InitialMagnification', 'fit');
title("Binary Image of test05");

%% Seventh Image
% Declaring and intialising the variable
% Using imread to read the given image and store it in a variable
Im = imread('task6_helipad.png');

% convolving the two objects together (Blurring the image using a
5x5 Gaussian Filter)
B = myconv(Im,kernel);

% Calculating the gradient of the blurred image in the x and y
direction
% using a 3x3 Sobel Filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Calculate the gradient magnitude
G = sqrt(Gx.^2 + Gy.^2);

% Calculating the gradient orientation
theta = atan2d(-Gy, Gx);

% Gradient orientations should be rounded to the nearest 45 degrees
theta = round(theta./45)*45;
theta(theta == 180) = -180;

% Performing non maxima suppression
low_threshold = 0.15;
high_threshold = 3.46;
binary_Image = myedge(Im, G, theta, low_threshold, high_threshold);

% Showing the gradient magnitude on screen as a greyscale image
```

```
figure(7);
imshow(binary_Image, 'InitialMagnification', 'fit');
title("Binary Image of helipad");
```

**Code for Task 6:**

Paste your code in here.

```
% Written by Nigel Tan Jin Chun
% Last Modified: 15/3/2023
% Name of the file: Lab1_Task6
% Purpose of the file: Find the center of the helipad and return the
row
% and column of the centre of the helipad

%% My Code Logic
% Load the image provided.
% Apply Gaussian blur to the image.
% Calculate the gradient magnitude and orientation of the blurred
image using Sobel filters.
% Apply non-maxima suppression to thin out the edges in the image.
% Apply hysteresis thresholding to identify edges with low and high
threshold values.
% Find contours in the image and filter out those that are not
likely to be the letter H.
% Calculate the centroid of the remaining contour to find the center
of the letter H.
% Plot an x at the center of the letter H.

clear all;close all;clc;

% Read the image
Im = imread('task6_helipad.png');

% Define the Gaussian filter and Sobel filter kernels
matrix = [2,4,5,4,2;4,9,12,9,4;5,12,15,12,5;4,9,12,9,4;2,4,5,4,2];
const = 1/159;
kernel = const .* matrix;
x_sobel = [-1 0 1; -2 0 2; -1 0 1];
y_sobel = [-1 -2 -1; 0 0 0; 1 2 1];

% Blur the image using the Gaussian filter
B = myconv(Im,kernel);

% Compute the gradient of the blurred image in the x and y direction
using the Sobel filter
Gx = myconv(B,x_sobel);
Gy = myconv(B,y_sobel);

% Compute the gradient magnitude and orientation
G = sqrt(Gx.^2 + Gy.^2);
theta = atan2d(-Gy, Gx);

% Round the gradient orientation to the nearest 45 degrees
theta = round(theta./45)*45;
theta(theta == 180) = -180;

% Perform non-maximum suppression to thin out the edges
% low_threshold = 0.25;
% high_threshold = 0.50;
% The low_threshold parameter sets the minimum threshold value for
an edge
% to be considered part of the output. Edges with gradient
magnitudes
% below this threshold are discarded.

% The high_threshold parameter sets the maximum threshold value for
% an edge to be considered part of the output, and is used to link
edges
% together into continuous curves. If an edge has a gradient
magnitude
% above the high threshold, it is considered a strong edge.
% If an edge has a gradient magnitude between the low and high
thresholds,
% it is considered a weak edge.
low_threshold = 0.25;
high_threshold = 1.2;
binary_Image = myedge(Im, G, theta, low_threshold, high_threshold);

% % Displaying the binary Image to check the edge detection
% figure(1);
% imshow(binary_Image);

% % Using the matlab built-in function to detect the binary image
% binary_Image2 = edge(Im,'canny');
% figure(2);
% imshow(binary_Image2);

% ---------------------------------------------------------------
% Apply a connected component analysis to detect the letter H
cc = bwconncomp(binary_Image);
stats = regionprops(cc, 'BoundingBox', 'Area');
areas = [stats.Area];
idx = find(areas == max(areas));

% Get the boundary of the largest connected component
b = bwboundaries(cc.PixelIdxList{idx});

% % Draw an X at the center of the boundary of the letter H
% x_center = round(mean(b{1}(:,2)));
% y_center = round(mean(b{1}(:,1)));
% marked_img = insertMarker(Im, [x_center y_center], 'x', 'size', 1,
'Color', 'red');

% Draw an X at the center of the bounding box of the letter H
% bbox(1) is the x coordinate of the top-left corner of the bounding
box,
% bbox(2) is the y coordinate of the top-left corner of the bounding
box,
% bbox(3) is the width of the bounding box, and
% bbox(4) is the height of the bounding box.
bbox = stats(idx).BoundingBox;
x_center = round(bbox(1)/1.3 + bbox(3)/2)-1;
y_center = round(bbox(2)/1.13 + bbox(4)/2);
```

```
% % Inserting the marker on the image
marked_img = insertMarker(Im, [x_center y_center], 'x', 'size',
1,'Color','red');

% Display the marked image
figure(1);
hold on
xlabel("x-coordinate");
ylabel("y-coordinate");
title("Center of the Helipad");
imshow(marked_img,[], 'InitialMagnification', 'fit');
axis on;

% Setting the limit of my axis
xlim([0, 100]);
ylim([0, 100]);

% Display the coordinates of the selected point below the marker
% text(x_center, y_center, ['(' num2str(x_center) ','
num2str(y_center) ')'], 'Color', 'r');
text(x_center, y_center+5, ['(' num2str(x_center) ','
num2str(y_center) ')'], 'Color', 'r', 'HorizontalAlignment',
'center');
hold off


% %% Comparing from the image given whether the letter x is located
% % % Inserting the marker on the image
% marked_img2 = insertMarker(Im, [67 51], 'x', 'size',
1,'Color','red');
%
% % Display the marked image
% figure(3);
% % title("Center of the Helipad");
% imshow(marked_img2, 'InitialMagnification', 'fit');
```