Contents

[Intelligent Vision System 2](#_Toc510380227)

[1. Introduction 2](#_Toc510380228)

[2. Methods and Results 2](#_Toc510380229)

[2.1. Fundamental Image Processing 2](#_Toc510380230)

[3. Static objects segmentation using edge detection 3](#_Toc510380231)

[3.1. Basic image processing techniques for decision making 8](#_Toc510380232)

[4. Background subtraction 10](#_Toc510380233)

[5. Object motion estimation and tracking 12](#_Toc510380234)

[5.1. Find corner points 12](#_Toc510380235)

[5.2. Face detection and tracking 16](#_Toc510380236)

[6. Discussion 16](#_Toc510380237)

[Subsection heading like this 16](#_Toc510380238)

[7. Conclusions 17](#_Toc510380239)

[References 17](#_Toc510380240)

[Appendix 18](#_Toc510380241)

# Intelligent Vision System

## Introduction

The aim of this lab is to understand basic image processing methods, e.g. image histogram, image enhancement. And at same the same time some algorithms are developed such as, decision making and face recognition and tracking in video processing.

Those algorithms are widely applied in autonomous systems including robotics, Unmanned Aerial Vehicles (UAVs), intelligent transportation systems, surveillance, augmented reality and virtual reality systems.

## Methods and Results

## Fundamental Image Processing

In this section basic operations on images of different types are implemented, such as image histograms and visualisation.

Figure 1 illustrates two images, which the one on the left is full of red colour and the other one are partly with yellow colour in the middle.

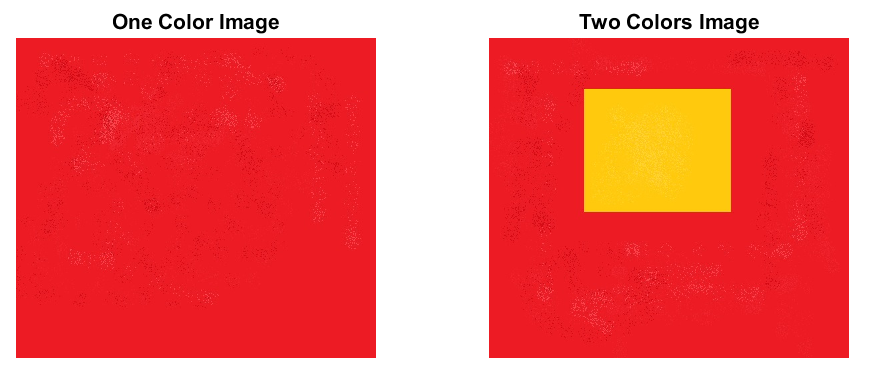


Figure 1 One colour image vs. Two colours images

Figure

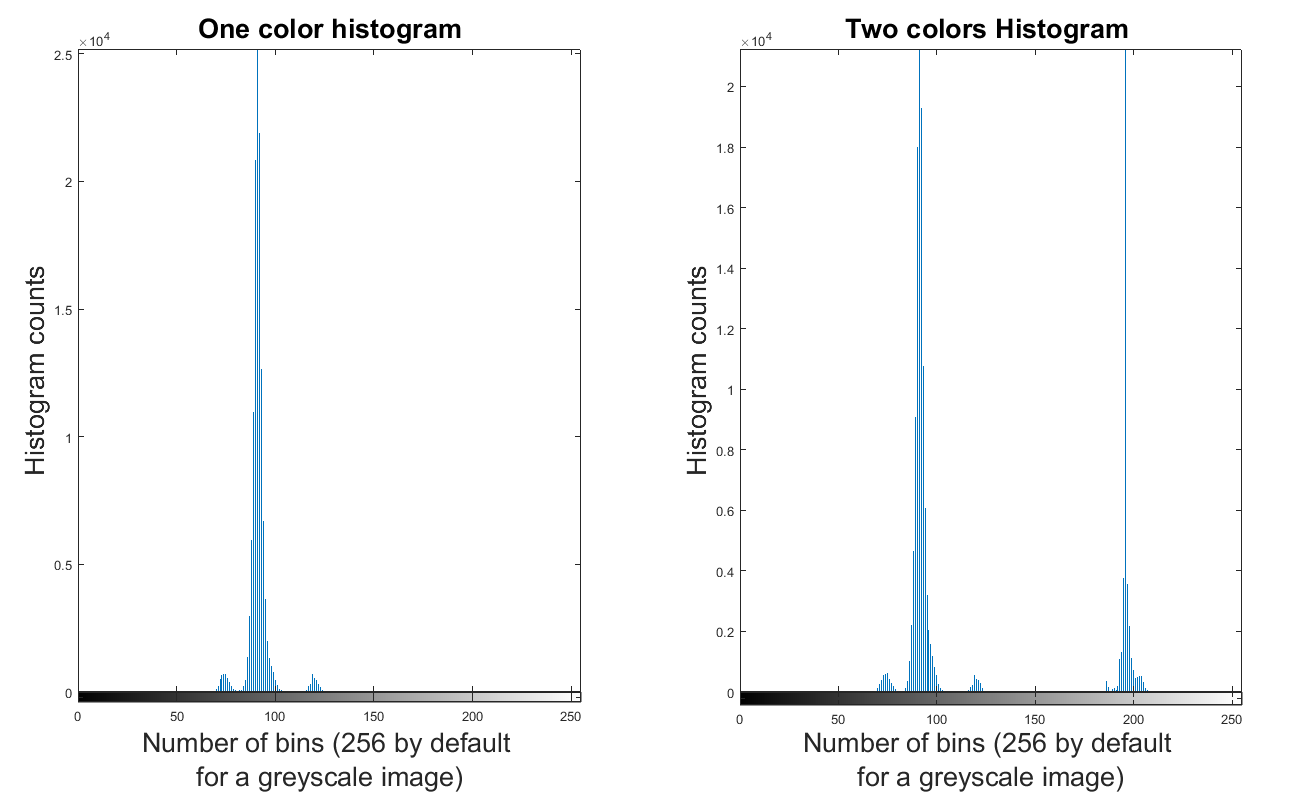


Figure 2 Histogram Comparison between one-color image and two-colour image

After converting the two images to gray format, figure 2 indicates that histogram of one color image’s values are mainly distributed around 100. But the one on the right side of figure 2, It clearly shows that the two-colour image contains some colour bins which are located around the value 200. It is possible to decide the image’s characters according to image histogram. As a result, it can be straightforward to distinguished the two images according to the histograms.

## Static objects segmentation using edge detection

In this section the basic image processing technology will be applied, for example, enhancing the contrast of the image, modelling different types of noise in the image, removing noise from the image, edge detecting and static object segmentation.

Figure 3 shows the original image and its histogram which concisely illustrates the intensity distribution of the original image. The histogram indicates that the majority values of each pixel are less than 175. And there is almost no value either less than 5 or more than 175. This image is underexposed.

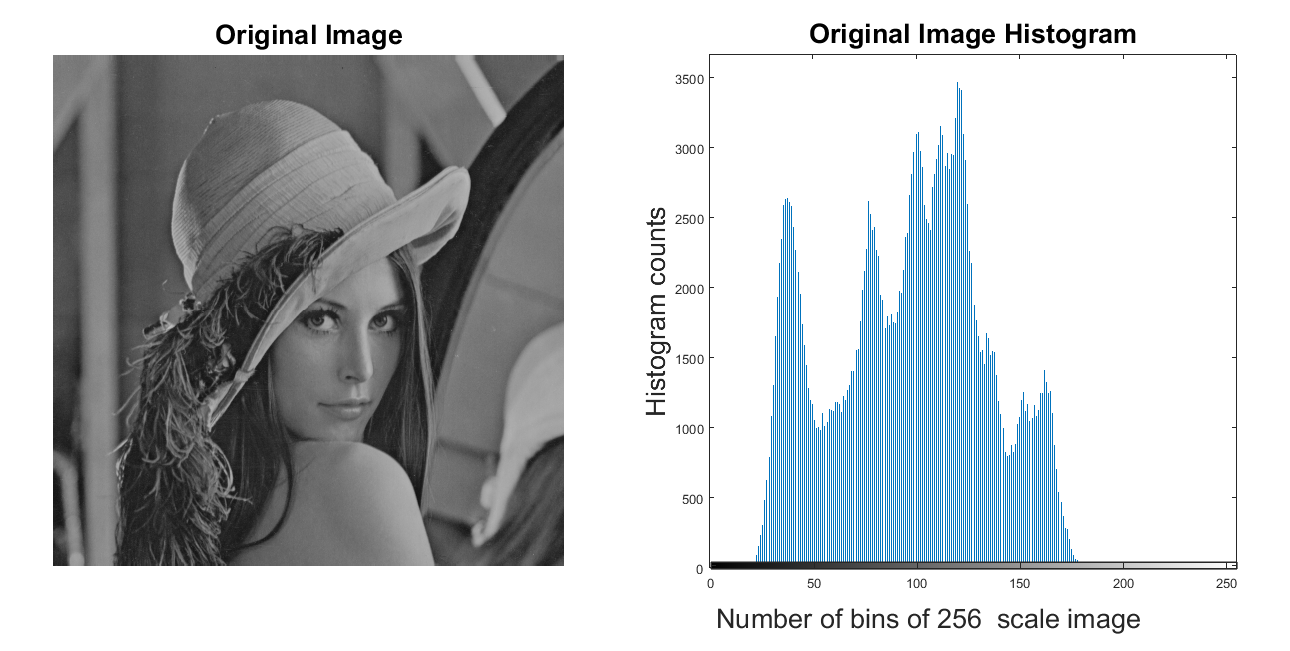


Figure 3 Original Image and its histogram

Figure4 shows the effect after dealing with histogram equalization. Pixels’ values are distributed uniformly in histogram. The image is more bringer than the original one. But the shape of histogram was changed.

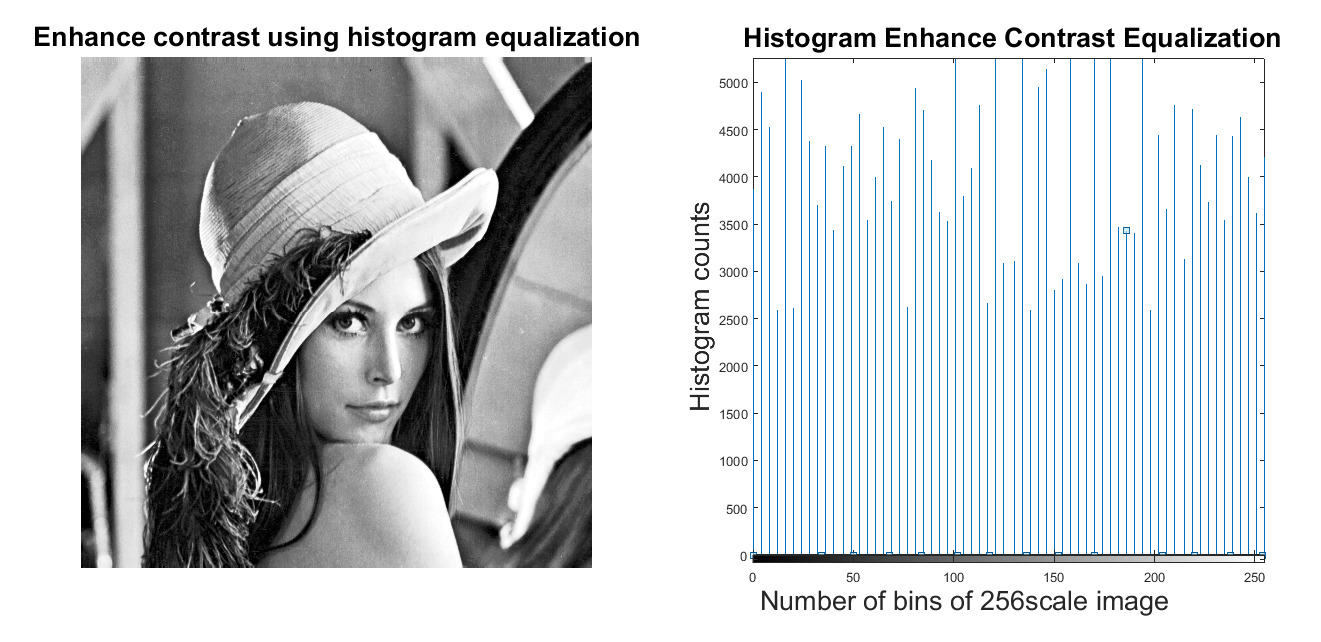


Figure 4 Applied histogram equalisation image and its histogram

Figure 5 shows the image after adjusting the image intensity values. Contrast of the image was adjusted. One hand image is brighter than the original one. On the other hand, all the pixels value are still kept the same shape in histogram as the original one.

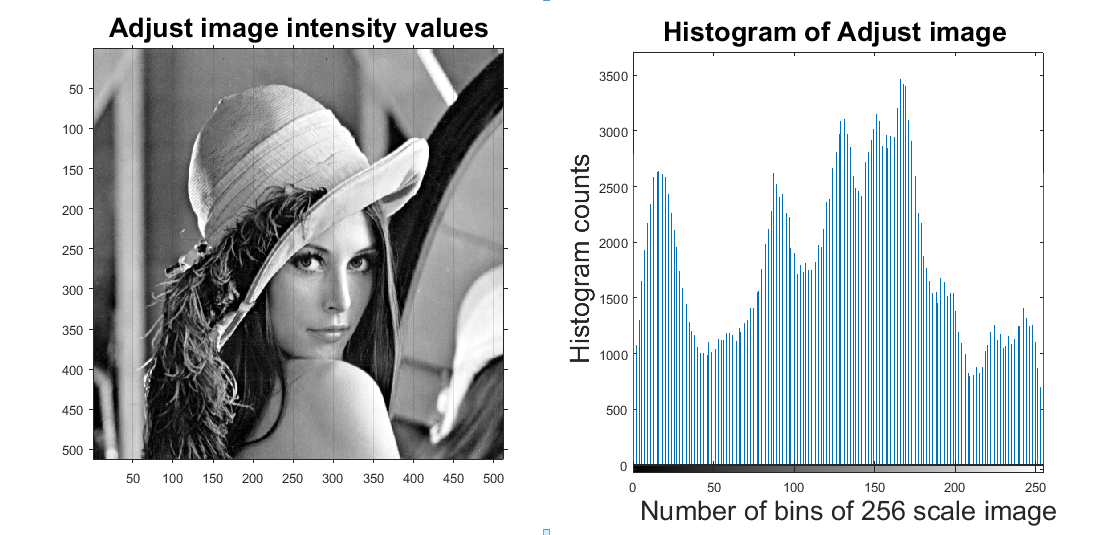


Figure 5 Applied gamma correction of the histogram image and its histogram

There are several methods for static objects segmentation such as Sobel, Canny and Prewill.

**Sobel operator**: Because of the weighted smoothing of the image, it has a certain ability to suppress the noise. But it cannot completely rule out the false edges that appear in the detection result. On the other hand, there exist some disadvantages of the Sobel method that it is sensitive to the noise. That is, it is sensitive to the noise. Figure 6 shows the different threshold value when apply Sobel operator. It shows that he magnitude of the edges will degrade as the level of noise present in image increases.

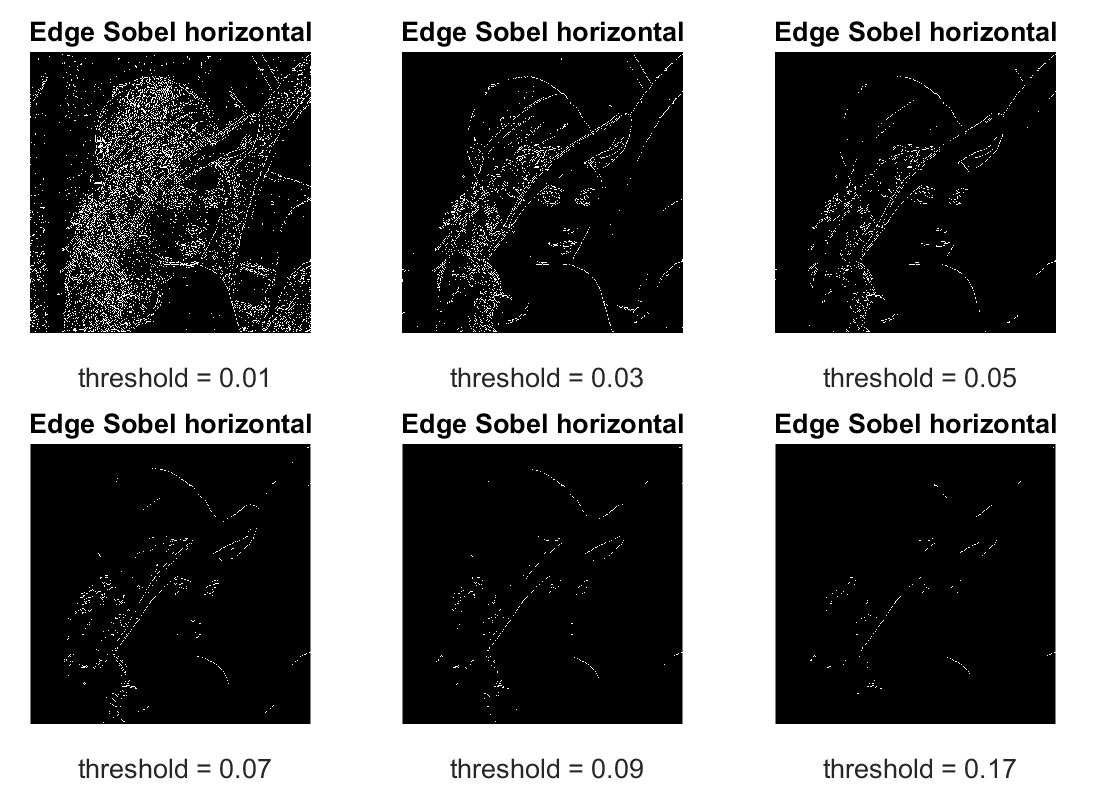


Figure 6 Edge detection with function sable

**Canny operator**: This operator uses a Gaussian function to smooth the image, so it has a strong ability to suppress noise. At the same time, it uses dual threshold detection and connection edges, and is better in multi-scale detection and directional search. The main disadvantage of Canny edge detector is that it is time consuming, due to its complex computation.

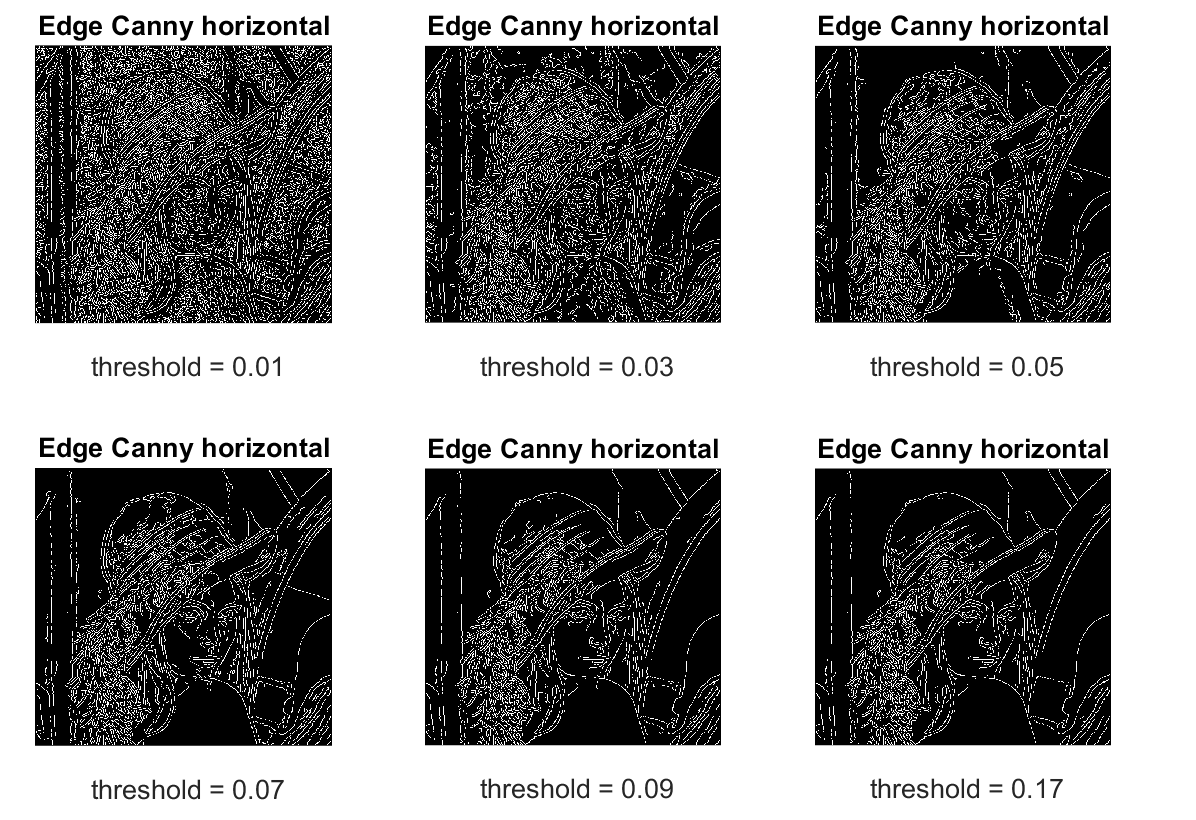


Figure 7 Edge detection with function canny.

The Prewitt masks are simpler to implement than the Sobel masks, but the slight computational difference between them typically is not an issue. The fact that the Sobel masks have better noise-suppression (smoothing)characteristics makes them preferable because, as mentioned in the previous section, noise suppression is an important issue when dealing with derivatives[1].

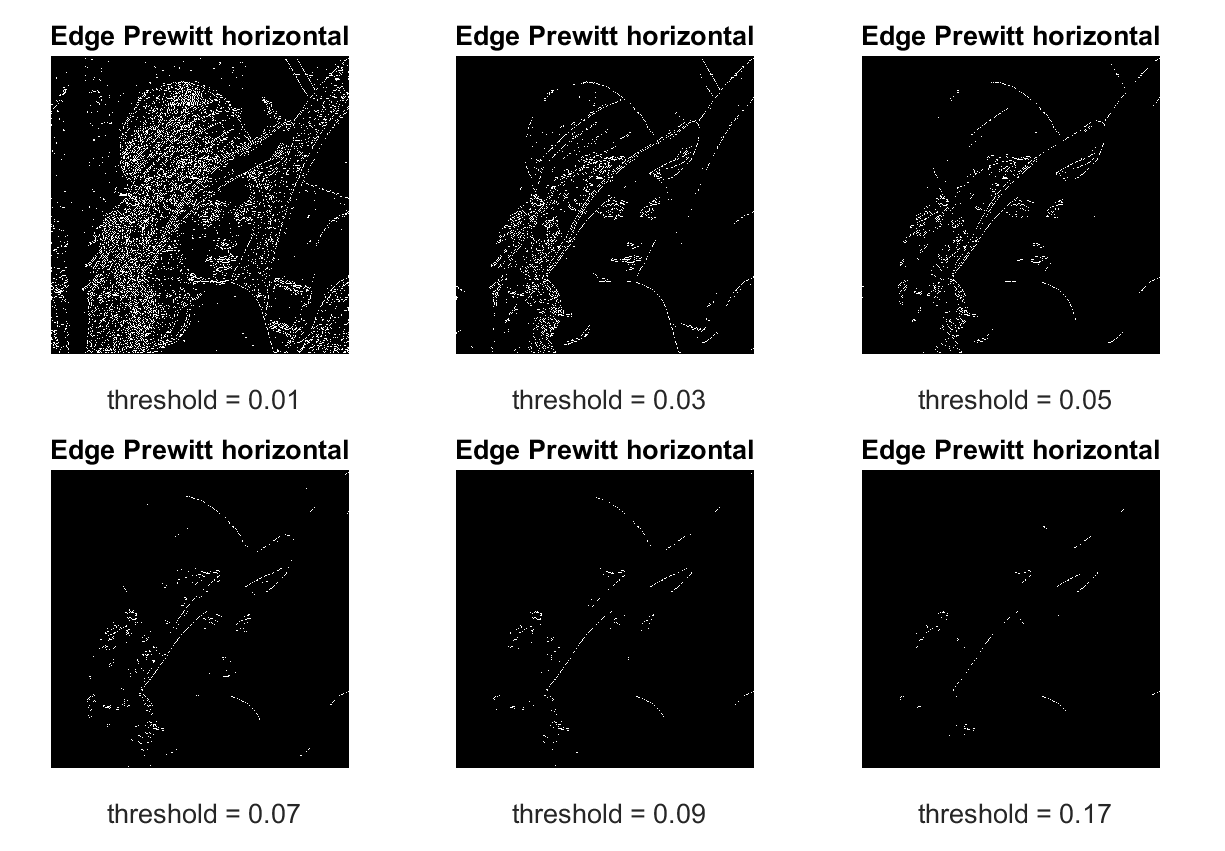


Figure 8 Prewitt

Table 1 Cooperation of strengths and weaknesses of detectors[2]

|  |  |  |
| --- | --- | --- |
| Operator | Strengths | Weaknesses |
| Sobel | Simple. Detects edges and their orientation | Inaccurate and sensitive to noise |
| Canny | Smoothing effect to remove noise. Good localization and response. Enhances signal to noise ratio. Immune to noisy environment. | Difficult to implement to reach real time response. Time consuming. |

## Basic image processing techniques for decision making

In this section

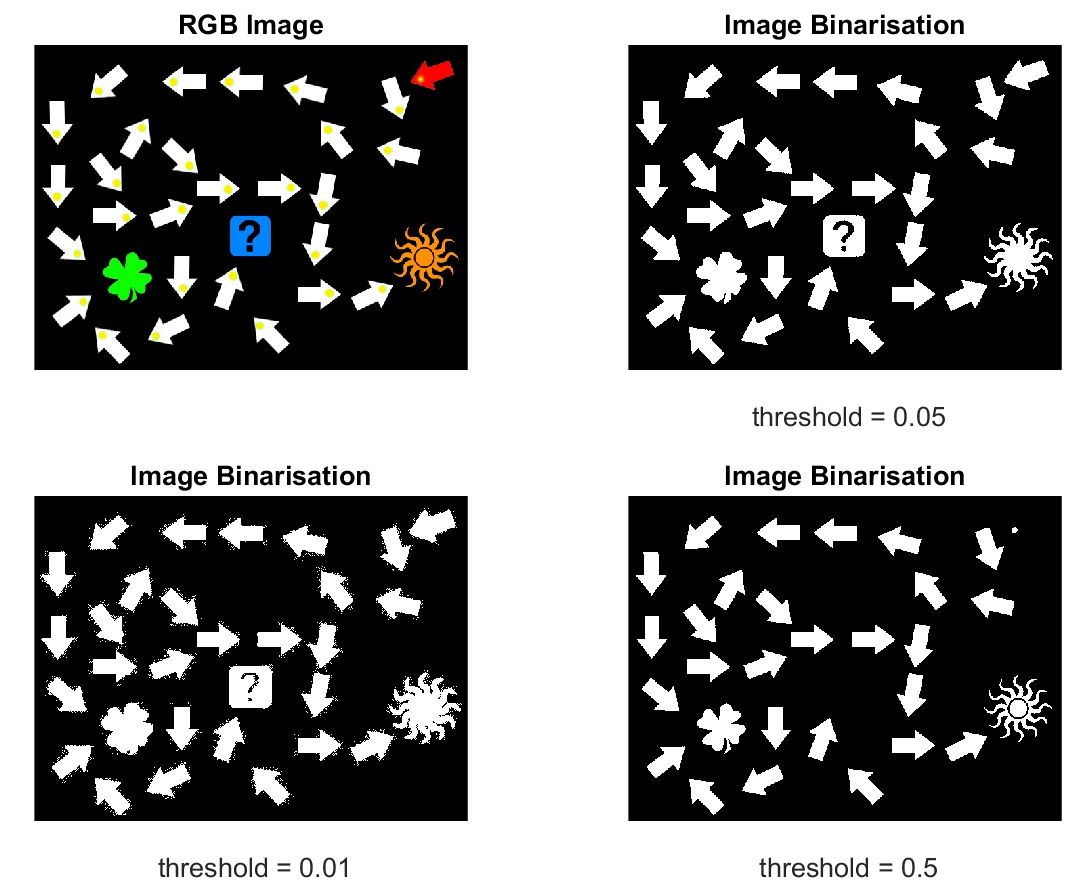


Figure 9 Image binarisation with different threshold

Figure 6 indicate the results of different threshold value’s affection in image binarisation. When the threshold value is set to 0.01, more than one thousand objects are found. It is far more than the objects in the original RGB image. When the threshold value was slightly turn to 0.05, the corresponded thirty-one objects were found in binarised image. With the threshold value was continued turning to ten times more than last value, the red arrow on the top-right was disappeared. After refine the parameter, threshold was set to 0.5 in order to represent all the thirty-one objects in image.

Function arrow\_finder -----Arrow/Non-arrow detection

As it is done in last part, bounding box were founded by using the function **bwlabel** and **regonprops.** Function **regionprops** is used to measure the properties of the specified area. After implementing this function, variable **props** is generated. Its data type is structure with three fields, Area, Centroid and BoundingBox. In filed, it will return a scalar that specifies the actual number of pixels in the region. According to this, it can be summarised that the all the arrows have the identified character which is less than 1700.This value is used to distinguish arrow and non-arrow objects in function arrow\_finder.m.

Function next\_object\_finder

After finding the red arrow, treasure finding path is initialised. To find the treasure in image, following steps are implemented for setting up the algorithm.

First of all, the central point of bounding box and central point of yellow area in each bounding box are extracted.

Secondly, one linear equation can be set up according to those two points for each bounding box. Arrow’s direction can be identified by a vector (x2-x1, y2-y1), which (x2, y2) represents central yellow point in the bounding box and (x1, y1) represents central point of the bounding box respectively.

Thirdly, the next object searching can be initialized according to the linear equation. Column will be increased or decreased according to the vector direction. At the same time, the pixel values of three consecutive points with a space of one are used as the basis for finding the next object. When the value of those three pixels for all channels When those three points’ pixel values are meet the corresponding threshold range, it indicated that the next target object has been found.

Lastly, the variable cur\_object will be assigned to main programme until the treasure was found.

## Background subtraction

Background subtraction, also known as foreground detection, is a technique in the fields of image processing and computer vision wherein an image's foreground is extracted for further processing. Generally, objects such as human, cars, text in foreground are the interesting region in image processing and detecting. In this section two main methods of background subtraction will be compared with different threshold.

**Motion detection base on Frame differencing algorithm**

Due to its low computational load, frame differencing algorithm is the simplest form of background subtraction. Moving objects are detected by the pixel-based values ‘changing.



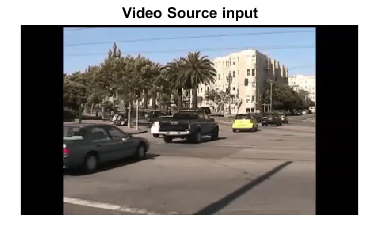
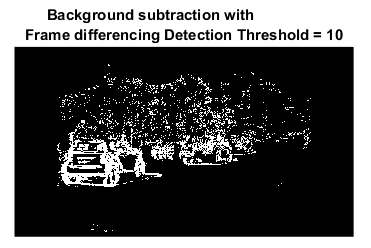
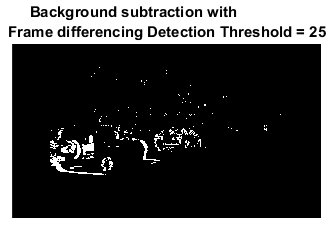


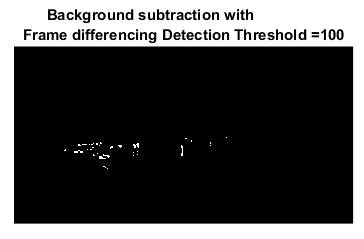
Figure 10 Motion detection

Table 2 Different value cooperation for Frame differencing algorithm

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Threshold** | **Tree** | **Passage** | **Cars** | **Car’s Shadow** |
| 10 | yes | yes | yes | yes |
| 25 | partial | yes | yes | Slightly |
| 50 | no | no | Yes but not clear enough | no |
| 100 | No | No | only some line | no |







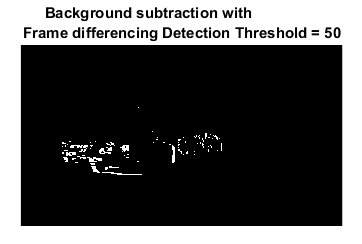


Figure 11 Background subtraction with different values

In this case, it is better to choose 25 as the threshold for the background subtraction.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Number of training** | **Number of Gaussian** | **Tree** | **Passage** | **Cars** | **Car’s Shadow** |
| 10 | 3 | yes | yes | yes | yes |
| 10 | 4 | yes | yes | yes |  |
| 10 | 5 |  |  |  |  |
|  |  |  |  |  |  |

Figure 12 Frame Difference

Gaussian mixture model algorithm for background subtraction

Figure 13 Gaussian mixture model algorithm for background subtraction

## Object motion estimation and tracking

In this section corner points will be implemented when optical flow algorithm was applied for an object tracking.

## Find corner points

Find corner point with function corner.

[C](file:///C:\Program%20Files\MATLAB\R2017a\help\images\ref\corner.html?browser=F1help#outputarg_C?browser=F1help)orner\_point = corner([I](file:///C:\Program%20Files\MATLAB\R2017a\help\images\ref\corner.html?browser=F1help#inputarg_I?browser=F1help)mage) detects corners in image and returns them in matrix Corner\_point.

After implemented function ***corner*** in Matlab, four points are found for image red\_square\_static.jpg.

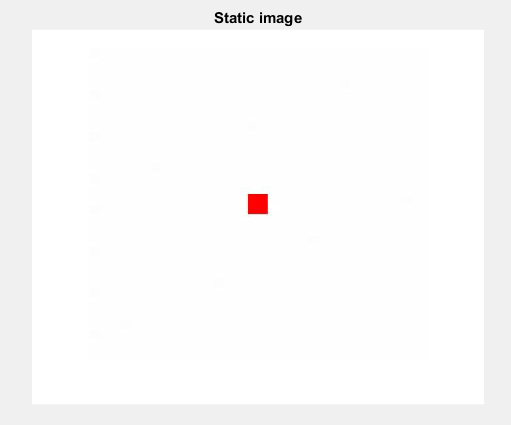


Figure 14 Corner points for red square

With image GingerBreadMan\_first.jpg, there are 200 corner points were found.

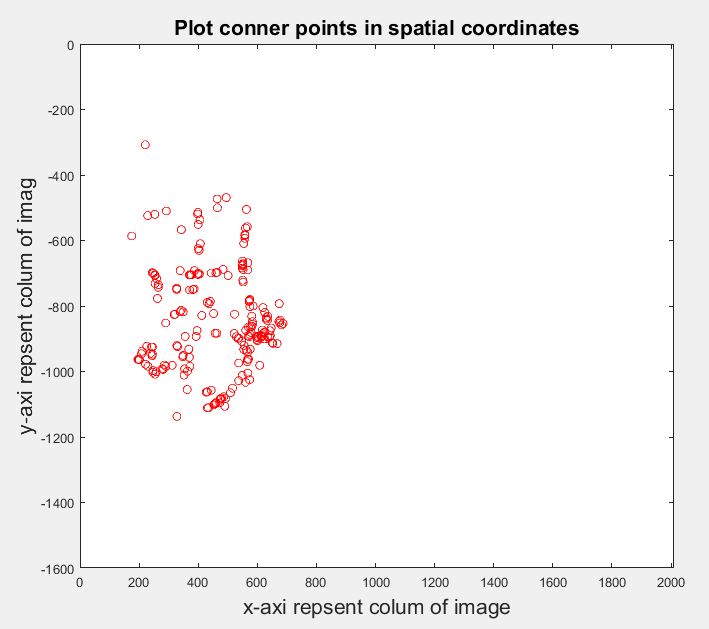
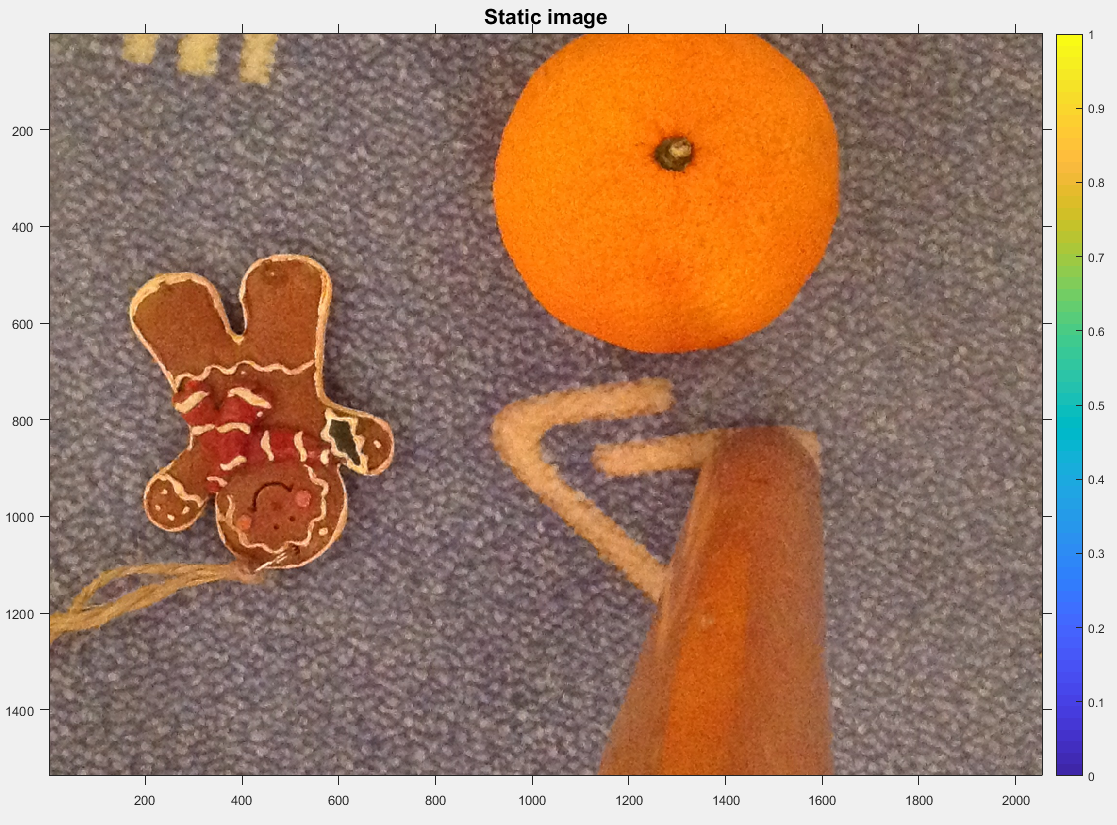


Figure 15 Corner points for image ginger man

Optical flow of the pixels finding

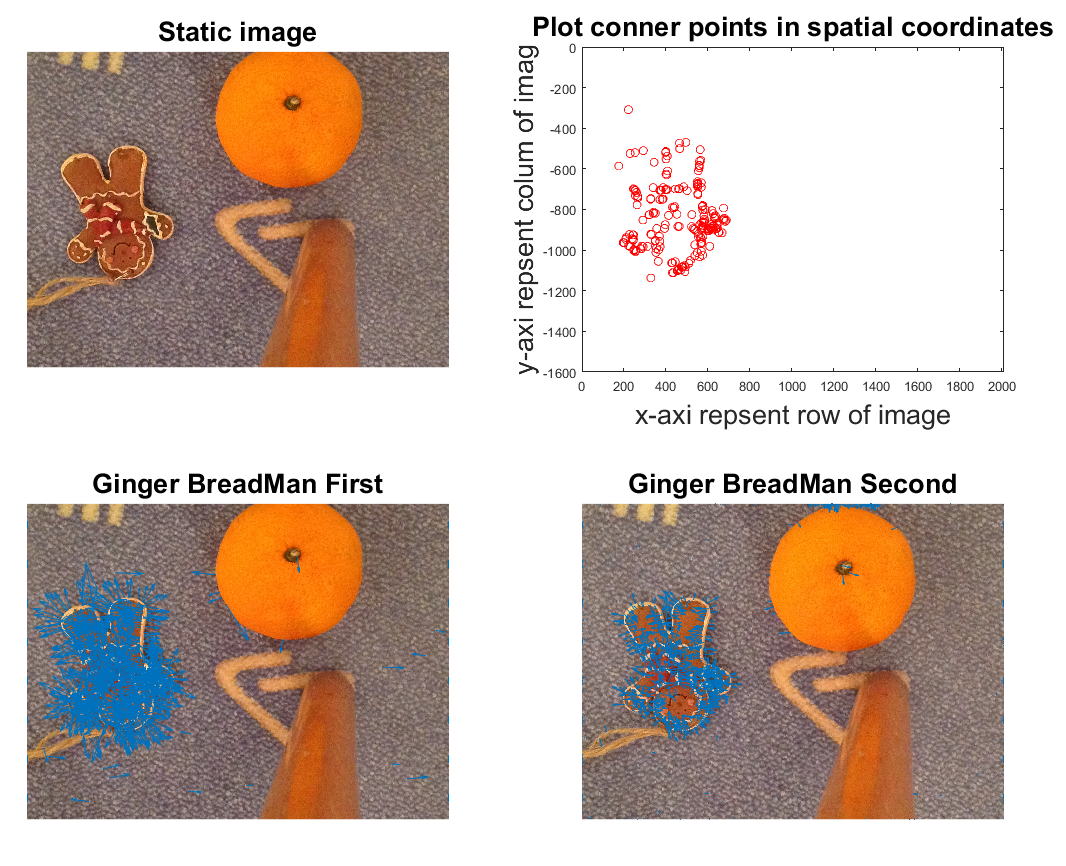


Figure 16 Optical Flo

Visualise the track on the last frame of the video

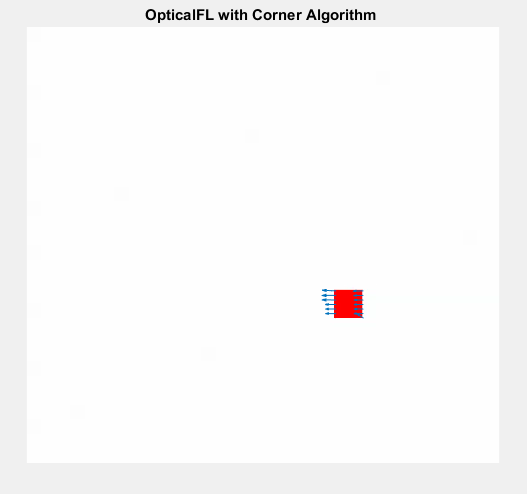


Figure 17 Last frame for track

estimated track by the optical flow algorithm in comparison with the ground truth values

The estimated track by the optical flow algorithm and the ground truth values are illustrated in figure 11.

Figure 13 shows that track point gotten by optical flow algorithm in each frame always exits gaps with the truth track, spatially at the very beginning.

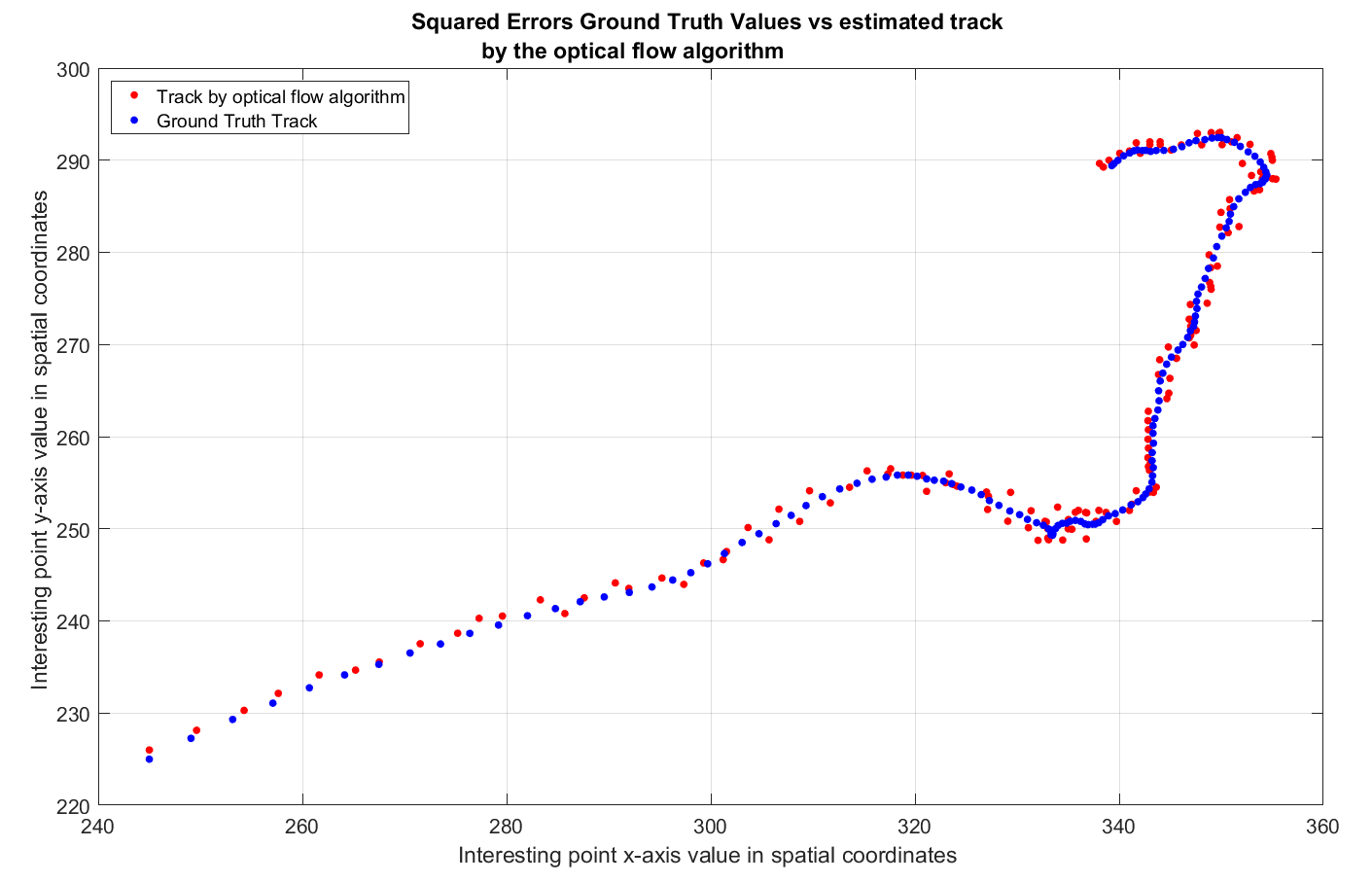


Figure 18 Truth and track points compilation

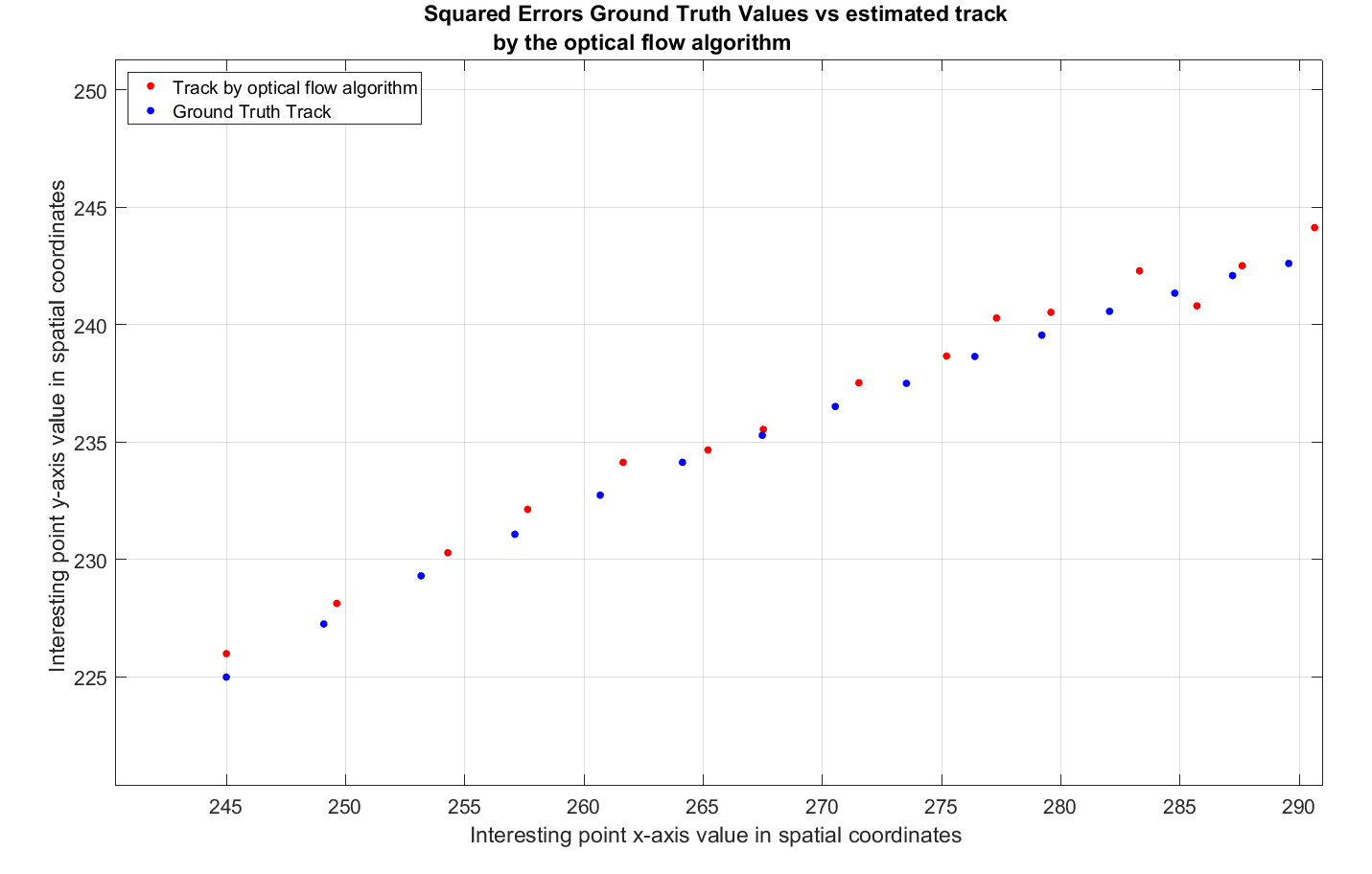


Figure 19 Ground Truth vs

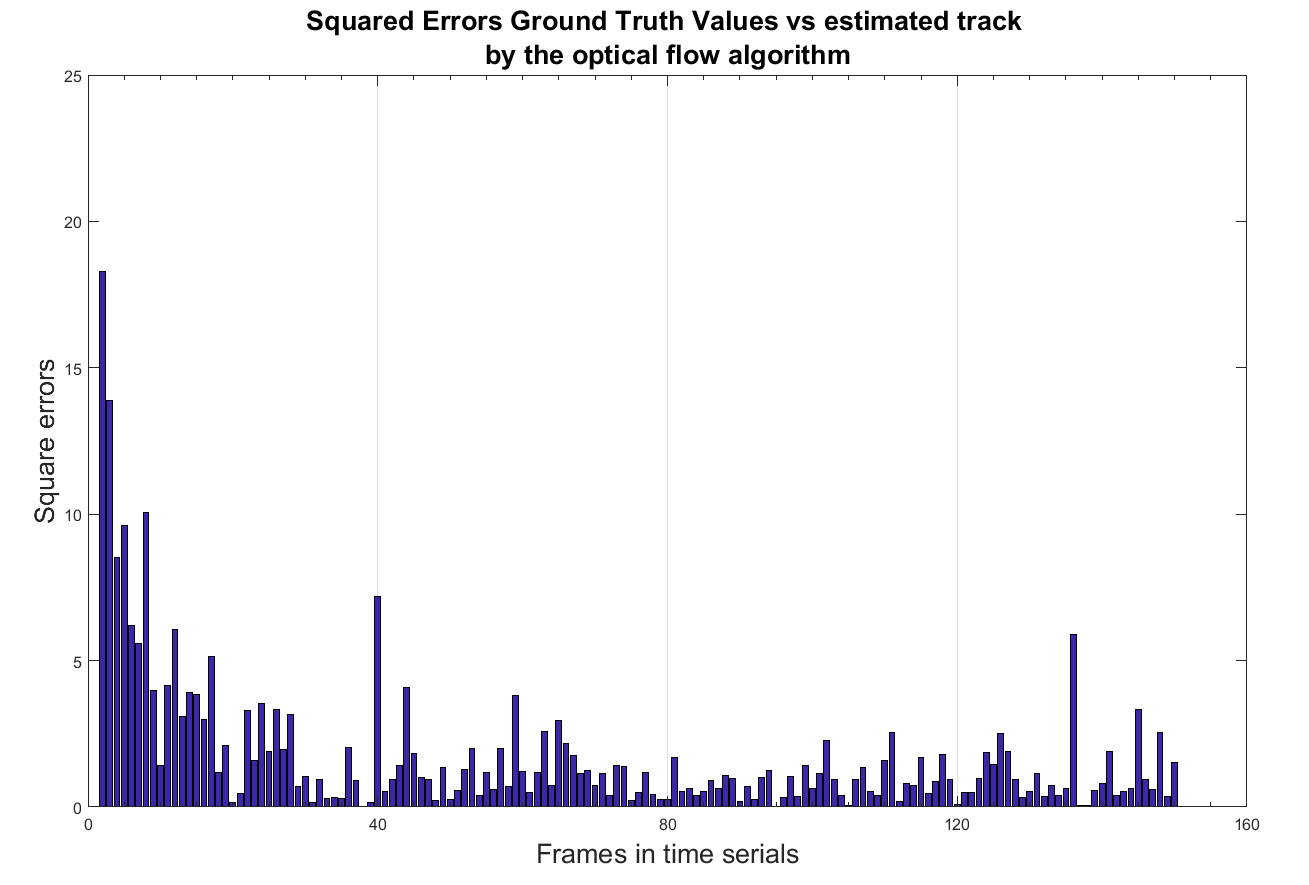


Figure 20 Square errors

## Face detection and tracking

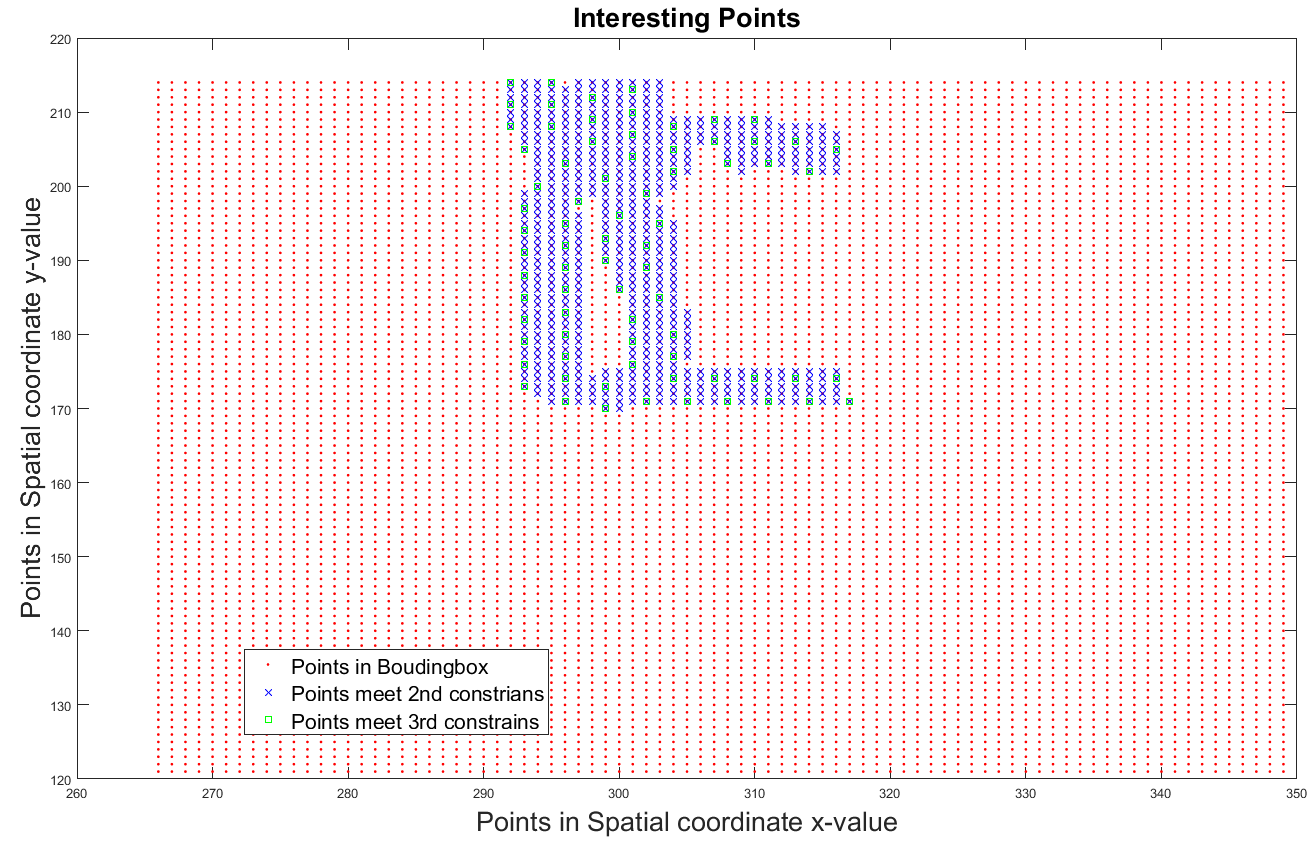


Figure 21 Interesting points

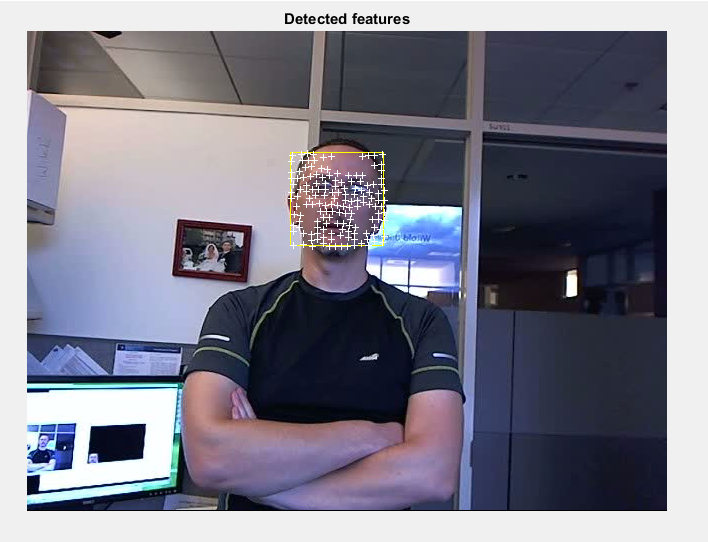


Figure 22 Face detection and tracking

## Conclusions

This report represents basic operations and algorithms in image processing. During the experiment, different thresholds were tested to choose the best fit one. To develop an algorithm for face detecting and tracking, Lucas-Kanade optical flow algorithm was used. Although the interesting points are extracted according to some constrains, the interesting points set were impossible to determine the direction of the motion unless the interesting points visible. And The visual information is only sufficient to determine the velocity in the direction orthogonal to the visible side of the rectangle.

### References

[1] R. C. Gonzalez, *Digital Image Processing*, Third Edit. United States of America: Upper Saddle River, 2008.

[2] D. Kim, “Sobel Operator and Canny Edge Detector ECE 480 Fall 2013 Team 4,” pp. 1–10, 2013.

## Appendix

Lab1

%% Labs total

clear all

close all

clc

%% lab1

%% Detection of an area of a predefined colour

% Color the duck yellow!

% Change the colour of the white pixels of an image to

% yellow on the image

im = imread('duckMallardDrake.jpg');

figure,imshow(im);

[nr,nc,np]= size(im);

newIm= zeros(nr,nc,np);

newIm= uint8(newIm);

for r= 1:nr

for c= 1:nc

if ( im(r,c,1)>180 && im(r,c,2)>180 && im(r,c,3)>180 )

% white feather of the duck; now change it to yellow

newIm(r,c,1)= 225;

newIm(r,c,2)= 225;

newIm(r,c,3)= 0;

else % the rest of the picture; no change

for p= 1:np

newIm(r,c,p)= im(r,c,p);

end

end

end

end

figure,imshow(newIm);

figure,

subplot(1,2,1);

imshow(im)

subplot(1,2,2);

imshow(newIm)

%% Find the pixels indexes with the yellow colour on the image ‘Two\_colour.jpg

im = imread('Two\_colour.jpg'); % read the image

figure;

imshow(im);

% extract RGB channels separatelly

red\_channel = im(:, :, 1);

green\_channel = im(:, :, 2);

blue\_channel = im(:, :, 3);

% label pixels of yellow colour

yellow\_map = green\_channel > 150 & red\_channel > 150 & blue\_channel < 50;

% extract pixels indexes

[i\_yellow, j\_yellow] = find(yellow\_map > 0);

% visualise the results

figure;

imshow(im); % plot the image

hold on;

scatter(j\_yellow, i\_yellow, 5, 'd') % highlighted the yellow pixels

%%

x = linspace(0,3\*pi,200);

y = cos(x) + rand(1,200);

subplot(1,2,1)

scatter(x,y);

x1 = linspace(0,3\*pi,100000);

y1 = cos(x1) + rand(1,100000);

subplot(1,2,2)

scatter(x1,y1,0.1)

%% Histogram difference between one-colour and two-color

clear all;

close all

clc;

%% Parameter declaration

Fontsize = 20;

Linewidth = 1;

%%

figure('Color',[1 1 1]);

subplot(1,2,1);

I1 = imread('One\_colour.jpg');

title ('One Color Image','Fontsize',Fontsize)

subplot(1,2,2)

I2 = imread('Two\_colour.jpg');

title ('Two Colors Image','Fontsize',Fontsize)

I1\_grey = rgb2gray(I1);

I2\_grey = rgb2gray(I2);

% visualise the results

figure('Color',[1 1 1]);

subplot(1,2,1);

imshow(I1); % plot the image

title ('One Color Image','Fontsize',Fontsize)

hold on;

subplot(1,2,2);

imshow(I2); % plot the image

title ('Two Colors Image','Fontsize',Fontsize)

hold on;

figure('Color',[1 1 1]);

subplot(1,2,1);imhist(I1\_grey);

xlabel('Number of bins (256 by default for a greyscale image)','Fontsize',Fontsize)

ylabel('Histogram counts','Fontsize',Fontsize)

title ('One color histogram','Fontsize',Fontsize)

subplot(1,2,2);imhist(I2\_grey);

xlabel('Number of bins (256 by default for a greyscale image)','Fontsize',Fontsize)

ylabel('Histogram counts','Fontsize',Fontsize)

title ('Two colors Histogram ','Fontsize',Fontsize)

xlabel('Number of bins (256 by default for a greyscale image)','Fontsize',Fontsize)

ylabel('Histogram counts','Fontsize',Fontsize)

h = imhist(I1\_grey);

h1 = h(1:10:256);

horz = 1:10:256;

figure, bar(horz,h1)

Lab2

%% lab2 edge detection

clear all

close all

clc

%% Parameter declaration

Fontsize = 20;

%%

I\_1 = imread('lena.gif');

figure('Color',[1 1 1]);

subplot(1,2,1);

imshow(I\_1);

title('Original Image','Fontsize',Fontsize);

subplot(1,2,2);

imhist(I\_1);

title('Image Histogram','Fontsize',Fontsize);

xlabel(['Number of bins of 256 ';'scale image '],'FontSize',20);

ylabel('Histogram counts','Fontsize',Fontsize)

% Compute an image histogram for the image (imhist). Visualise the results. Analysing the

% histogram think about the best way of enhancement the image

imfinfo lena.gif;

% figure;

% imhist(I\_1);

% Apply the histogram equalisation to the image (histeq). Visualise the results. Compute an image

% histogram for the corrected image. Visualise the results. Compare it with the original histogram.

% Histeq image

figure('Color',[1 1 1]);

subplot(1,2,1);

I\_eq = histeq(I\_1);

imshow(I\_eq);

title('Enhance contrast using histogram equalization','Fontsize',Fontsize);

subplot(1,2,2);

imhist(I\_eq);

title('Enhance contrast using histogram equalization','Fontsize',Fontsize);

xlabel(['Number of bins of 256 ';'scale image '],'FontSize',20);

ylabel('Histogram counts','Fontsize',Fontsize)

% Imadjust image

figure('Color',[1 1 1]);

subplot(1,2,1);

I\_adj =imadjust(I\_1);

imshow(I\_adj);

title('Adjust image intensity values','Fontsize',Fontsize);

subplot(1,2,2);

imhist(I\_adj);

title('Histogram of Adjust image ','Fontsize',Fontsize);

xlabel(['Number of bins of 256 ';'scale image '],'FontSize',20);

ylabel('Histogram counts','Fontsize',Fontsize)

check\_point = 8;

%% Apply the gamma correction of the histogram to the image

%% Synthesise two images from the image ¡®lena.gif¡¯ with two types of noise

% Gaussian white noise with constant mean and variance

I\_gau = imnoise(I\_1,'gaussian',0,0.01);

% figure,

% imshow(I\_gau)

% Synthesise two images

% hold on;

% J = imnoise(I,'salt & pepper',d) adds salt and pepper noise to the image I, where d is the

% noise density. This affects approximately d\*numel(I) pixels. The default for d is 0.05.

I\_sp = imnoise(I\_1,'salt & pepper',0.05);

% imshow(I\_sp)

I\_gau\_gauf = imgaussfilt(I\_gau);

% figure;

% imshow(I\_gau\_gauf);

I\_sp\_gauf = imgaussfilt(I\_sp);

figure('Color',[1 1 1]);

imshow(I\_sp\_gauf);

title('I\_sp\_gauf')

% Apply the median filter

I\_sp\_med = medfilt2(I\_sp);

figure;

imshow(I\_sp\_med);

title('Aalt and Pepper median filter','Fontsize',Fontsize)

% lab2 9-11

% BW = edge(I\_1,'Sobel',threshold)

% Find edges on the image ¡®lena.gif¡¯ with the Sobel operator (edge(¡­, ¡sobel¡, ¡­)).

% Vary the threshold parameter value and draw conclusions about its influence

% over the quality of the segmented image. Visualise the results with the optimal threshold value;

% Sobel

thteshold1 = 0.01 ;

thteshold2 = 0.03 ;

thteshold3 = 0.05 ;

thteshold4 = 0.07 ;

thteshold5 = 0.09 ;

thteshold6 = 0.11 ;

%%

figure('Color',[1 1 1]);

I\_edge\_s = edge(I\_1, 'Sobel',thteshold1,'horizontal'); % threshold value

subplot(2,3,1);

imshow(I\_edge\_s);

title('Edge Sobel horizontal','Fontsize',Fontsize)

xlabel(' threshold = 0.01','Fontsize',Fontsize )

I\_edge\_s = edge(I\_1, 'Sobel',thteshold2,'horizontal'); % threshold value

subplot(2,3,2);

imshow(I\_edge\_s);

title('Edge Sobel horizontal','Fontsize',Fontsize)

xlabel(' threshold = 0.03','Fontsize',Fontsize )

I\_edge\_s = edge(I\_1, 'Sobel',thteshold3,'horizontal'); % threshold value

subplot(2,3,3);

imshow(I\_edge\_s);

title('Edge Sobel horizontal','Fontsize',Fontsize)

xlabel(' threshold = 0.05' ,'Fontsize',Fontsize)

I\_edge\_s = edge(I\_1, 'Sobel',thteshold4,'horizontal'); % threshold value

subplot(2,3,4);

imshow(I\_edge\_s);

title('Edge Sobel horizontal','Fontsize',Fontsize)

xlabel(' threshold = 0.07' ,'Fontsize',Fontsize)

I\_edge\_s = edge(I\_1, 'Sobel',thteshold5,'horizontal'); % threshold value

subplot(2,3,5);

imshow(I\_edge\_s);

title('Edge Sobel horizontal','Fontsize',Fontsize)

xlabel(' threshold = 0.09','Fontsize',Fontsize )

I\_edge\_s = edge(I\_1, 'Sobel',thteshold6,'horizontal'); % threshold value

subplot(2,3,6);

imshow(I\_edge\_s);

title('Edge Sobel horizontal','Fontsize',Fontsize)

xlabel(' threshold = 0.17','Fontsize',Fontsize )

%% Canny

figure('Color',[1 1 1]);

I\_edge\_s = edge(I\_1, 'Canny',thteshold1,'horizontal'); % threshold value

subplot(2,3,1);

imshow(I\_edge\_s);

title('Edge Canny horizontal','Fontsize',Fontsize)

xlabel(' threshold = 0.01','Fontsize',Fontsize )

I\_edge\_s = edge(I\_1, 'Canny',thteshold2,'horizontal'); % threshold value

subplot(2,3,2);

imshow(I\_edge\_s);

title('Edge Canny horizontal','Fontsize',Fontsize)

xlabel(' threshold = 0.03','Fontsize',Fontsize )

I\_edge\_s = edge(I\_1, 'Canny',thteshold3,'horizontal'); % threshold value

subplot(2,3,3);

imshow(I\_edge\_s);

title('Edge Canny horizontal','Fontsize',Fontsize)

xlabel(' threshold = 0.05' ,'Fontsize',Fontsize)

I\_edge\_s = edge(I\_1, 'Canny',thteshold4,'horizontal'); % threshold value

subplot(2,3,4);

imshow(I\_edge\_s);

title('Edge Canny horizontal','Fontsize',Fontsize)

xlabel(' threshold = 0.07' ,'Fontsize',Fontsize)

I\_edge\_s = edge(I\_1, 'Canny',thteshold5,'horizontal'); % threshold value

subplot(2,3,5);

imshow(I\_edge\_s);

title('Edge Canny horizontal','Fontsize',Fontsize)

xlabel(' threshold = 0.09','Fontsize',Fontsize )

I\_edge\_s = edge(I\_1, 'Canny',thteshold6,'horizontal'); % threshold value

subplot(2,3,6);

imshow(I\_edge\_s);

title('Edge Canny horizontal','Fontsize',Fontsize)

xlabel(' threshold = 0.17','Fontsize',Fontsize )

%% Prewitt

figure('Color',[1 1 1]);

I\_edge\_s = edge(I\_1, 'Prewitt',thteshold1,'horizontal'); % threshold value

subplot(2,3,1);

imshow(I\_edge\_s);

title('Edge Prewitt horizontal','Fontsize',Fontsize)

xlabel(' threshold = 0.01','Fontsize',Fontsize )

I\_edge\_s = edge(I\_1, 'Prewitt',thteshold2,'horizontal'); % threshold value

subplot(2,3,2);

imshow(I\_edge\_s);

title('Edge Prewitt horizontal','Fontsize',Fontsize)

xlabel(' threshold = 0.03','Fontsize',Fontsize )

I\_edge\_s = edge(I\_1, 'Prewitt',thteshold3,'horizontal'); % threshold value

subplot(2,3,3);

imshow(I\_edge\_s);

title('Edge Prewitt horizontal','Fontsize',Fontsize)

xlabel(' threshold = 0.05' ,'Fontsize',Fontsize)

I\_edge\_s = edge(I\_1, 'Prewitt',thteshold4,'horizontal'); % threshold value

subplot(2,3,4);

imshow(I\_edge\_s);

title('Edge Prewitt horizontal','Fontsize',Fontsize)

xlabel(' threshold = 0.07' ,'Fontsize',Fontsize)

I\_edge\_s = edge(I\_1, 'Prewitt',thteshold5,'horizontal'); % threshold value

subplot(2,3,5);

imshow(I\_edge\_s);

title('Edge Prewitt horizontal','Fontsize',Fontsize)

xlabel(' threshold = 0.09','Fontsize',Fontsize )

I\_edge\_s = edge(I\_1, 'Prewitt',thteshold6,'horizontal'); % threshold value

subplot(2,3,6);

imshow(I\_edge\_s);

title('Edge Prewitt horizontal','Fontsize',Fontsize)

xlabel(' threshold = 0.17','Fontsize',Fontsize )

%%

% I\_edge\_ca = edge(I\_1, 'Canny',([]),'vertical'); % threshold value

% subplot(2,3,2);

% imshow(I\_edge\_s);

% title('Edge Canny vertical')

%

%

% I\_edge\_pr = edge(I\_1, 'Prewitt',([]),'both'); % threshold value

% subplot(2,3,3);

% imshow(I\_edge\_s);

% title('Edge Prewitt both')

Lab3

close all;

clear all;

clc

%% Reading image

im = imread('Treasure\_medium.jpg'); % change name to process other images

imshow(im);

%% Binarisation

bin\_threshold = 0.05; % parameter to vary

bin\_im = im2bw(im, bin\_threshold);

imshow(bin\_im);

%% Extracting connected components

con\_com = bwlabel(bin\_im);

imshow(label2rgb(con\_com));

%% Computing objects properties

props = regionprops(con\_com);

%% Drawing bounding boxes

n\_objects = numel(props);

figure;

imshow(im);

hold on;

for object\_id = 1 : n\_objects

rectangle('Position', props(object\_id).BoundingBox, 'EdgeColor', 'b');

end

hold off;

%% Arrow/non-arrow determination

arrow\_ind = arrow\_finder();

%% build data structure Bounding box data sets AND Yellow area data sets,

Yellow\_matrix = zeros(0,2);

Yellow\_dataset = zeros(0,2);

iteration =0;

for i = 1:length(props)

box\_x = round(props(i).BoundingBox(1));

box\_y = round(props(i).BoundingBox(2));

x\_length= round(props(i).BoundingBox(3));

y\_length= round(props(i).BoundingBox(4));

Boundbox\_dataset= zeros(0,2);

for c = box\_x :box\_x+x\_length

for r = box\_y :box\_y+y\_length

Boundbox\_dataset = [Boundbox\_dataset;[c,r]];

if ( im(r,c,1)>250 && im(r,c,2)>230 && im(r,c,3)<50 )

% Yellow\_matrix = [Yellow\_matrix;[c,r]];

end

end

if ismember(i,arrow\_ind)

m =1;

else

m =0;

end

iteration = iteration +1;

% Yellow\_dataset = [Yellow\_matrix;[c,r]];

% yellow\_cid = mean(Yellow\_matrix);

% Bounding\_box\_stru(i) = struct('Index',i,'Matrix',Boundbox\_dataset,'Yellow\_matrix',Yellow\_matrix,'Yellow\_cid',yellow\_cid,'Is\_arrow',m);

Bounding\_box\_stru(i) = struct('Index',i,'Matrix',Boundbox\_dataset,'Is\_arrow',m);

end

end

%% Finding red arrow

n\_arrows = numel(arrow\_ind);

start\_arrow\_id = 0;

% check each arrow until find the red one

for arrow\_num = 1 : n\_arrows

object\_id = arrow\_ind(arrow\_num); % determine the arrow id

% extract colour of the centroid point of the current arrow

centroid\_colour = im(round(props(object\_id).Centroid(2)), round(props(object\_id).Centroid(1)), :);

if centroid\_colour(:, :, 1) > 240 && centroid\_colour(:, :, 2) < 10 && centroid\_colour(:, :, 3) < 10

% the centroid point is red, memorise its id and break the loop

start\_arrow\_id = object\_id;

break;

end

end

%% Detection of yellow area

cur\_object = start\_arrow\_id; % start from the red arrow

path = cur\_object;

for i = 1: length(props)

Yellow\_matrix = zeros(0,2);

yellow\_cid = zeros(0,2);

box\_x = round(props(i).BoundingBox(1));

box\_y = round(props(i).BoundingBox(2));

x\_length= round(props(i).BoundingBox(3));

y\_length= round(props(i).BoundingBox(4));

for c = box\_x: box\_x + x\_length

for r = box\_y: box\_y+y\_length

if ( im(r,c,1)>250 && im(r,c,2)>230 && im(r,c,3)<50 )

Yellow\_matrix = [Yellow\_matrix;[c,r]];

yellow\_cid = mean(Yellow\_matrix);

% text (X\_bd,Y\_bd,'arrow','color','black')

end

iteration = iteration +1;

end

Yellow\_matrix\_stru(i) = struct('Index',i,'Matrix',Yellow\_matrix,'Yellow\_cid',yellow\_cid);

end

end

% Central point of yellow area

yellow\_cid = mean(Yellow\_matrix);

x\_yellow\_cid =yellow\_cid(1);

y\_yellow\_cid =-yellow\_cid(2);

y\_cid = props(cur\_object).Centroid(2);

x\_cid = props(cur\_object).Centroid(1);

k1 = (y\_yellow\_cid - y\_cid)/(x\_yellow\_cid-x\_cid);

y\_s = round(k1\* props(path).BoundingBox(1));

intercept = y\_yellow\_cid - k1\*x\_yellow\_cid;

%% plot arrow in spatial

% figure;

%

% P1\_x = Bounding\_box\_stru.Matrix;

% plot(P1\_x(:,1),-P1\_x(:,2),'rx');

% axis([0 640 -480 0])

% hold on

% P1\_cx =props(1).Centroid

% plot(P1\_cx(1),-P1\_cx(2),'bo')

% hold on

% P1\_yx = Yellow\_matrix\_stru.Yellow\_cid

% plot(P1\_yx(1),-P1\_yx(2),'go')

% hold on

%

% P2\_x = Bounding\_box\_stru(2).Matrix;

% plot(P2\_x(:,1),-P2\_x(:,2),'rx');

% axis([0 640 -480 0])

% hold on

% P2\_cx =props(2).Centroid

% plot(P2\_cx(1),-P2\_cx(2),'bo')

% hold on

% P2\_yx = Yellow\_matrix\_stru(2).Yellow\_cid

% plot(P2\_yx(1),-P2\_yx(2),'go')

% hold on

%

% kp = (-P1\_cx(2)+ P1\_yx(2))/(P1\_cx(1)-P1\_yx(1));

% p\_intercept = -P1\_yx(2)- kp\*P1\_yx(1);

% p\_x= 0:1:100;

% p\_y = kp\*p\_x + p\_intercept

% plot(p\_x,p\_y,'k-')

% check\_point = 8;

%% Hunting

cur\_object = start\_arrow\_id; % start from the red arrow

path = cur\_object;

% while the current object is an arrow, continue to search

while ismember(cur\_object, arrow\_ind)

% You should develop a function next\_object\_finder, which returns

% the ID of the nearest object, which is pointed at by the current

% arrow. You may use any other parameters for your function.

cur\_object = next\_object\_finder(cur\_object);

path(end + 1) = cur\_object;

end

%% visualisation of the path

imshow(im);

hold on;

for path\_element = 1 : numel(path) - 1

object\_id = path(path\_element); % determine the object id

rectangle('Position', props(object\_id).BoundingBox, 'EdgeColor', 'y');

str = num2str(path\_element);

text(props(object\_id).BoundingBox(1), props(object\_id).BoundingBox(2), str, 'Color', 'r', 'FontWeight', 'bold', 'FontSize', 14);

end

% visualisation of the treasure

treasure\_id = path(end);

rectangle('Position', props(treasure\_id).BoundingBox, 'EdgeColor', 'g');

lab3 Function arrow\_finder()

function [arrow\_ind] = arrow\_finder()

% Distinguish arrow and non-arrow objects

% Call variable object\_id, props from main program

object\_id = evalin('base','object\_id');

props = evalin('base','props');

arrow\_ind = zeros(0,1);

for object\_id = 1: length(props)

if (props(object\_id).Area > 1700)

text (props(object\_id).BoundingBox(1), props(object\_id).BoundingBox(2),'not arrow','color','blue','FontSize',14);

else

arrow\_ind = [arrow\_ind ;object\_id];

str = num2str(object\_id);

text( props(object\_id).BoundingBox(1), props(object\_id).BoundingBox(2),str,'color','blue','FontSize',14);

end

end

end

Lab3 function next\_object\_finder

function cur\_object = next\_object\_finder( cur\_object )

%% variable from main programe

% cur\_object = evalin('base','cur\_object');

% start\_arrow\_id = evalin('base','start\_arrow\_id');

% object\_id = evalin('base','object\_id');

props = evalin('base','props');

% k1 = evalin('base','k1');

arrow\_ind = zeros(0,1);

arrow\_ind = evalin('base','arrow\_ind');

%% Reading image

im = imread('Treasure\_medium.jpg'); % change name to process other images

imshow(im);

%% Binarisation

bin\_threshold = 0.05; % parameter to vary

bin\_im = im2bw(im, bin\_threshold);

imshow(bin\_im);

%% Extracting connected components

con\_com = bwlabel(bin\_im);

imshow(label2rgb(con\_com));

%% Computing objects properties

props = regionprops(con\_com);

%% building boudingbox data set

for i = 1:length(props)

box\_x = round(props(i).BoundingBox(1));

box\_y = round(props(i).BoundingBox(2));

x\_length= round(props(i).BoundingBox(3));

y\_length= round(props(i).BoundingBox(4));

Boundbox\_dataset= zeros(0,2);

for j = box\_x :box\_x+x\_length

for k = box\_y :box\_y+y\_length

Boundbox\_dataset = [Boundbox\_dataset;[j,k]];

end

if ismember(i,arrow\_ind)

m =1;

else

m =0;

end

Bounding\_box\_stru(i) = struct('Index',i,'Matrix',Boundbox\_dataset,'Is\_arrow',m);

end

end

%% Detection of yellow area

% cur\_object = start\_arrow\_id; % start from the red arrow

path = cur\_object;

yellow\_matrix = zeros(0,2);

x\_length = round(props(cur\_object).BoundingBox(3));

y\_length = round(props(cur\_object).BoundingBox(4));

iteration =0;

for i = 1:y\_length

for j = 1:x\_length

c = round(props(cur\_object).BoundingBox(1))-1+j;

r = round(props(cur\_object).BoundingBox(2))-1+i;

if (im(r,c,1)>220 && im(r,c,2)>67 && im(r,c,3)<40)

yellow\_matrix = [yellow\_matrix;[c,r]];

% text (X\_bd,Y\_bd,'arrow','color','black')

end

iteration = iteration +1;

end

end

% Central point of yellow area

yellow\_cid = mean(yellow\_matrix);

x\_yellow\_cid =yellow\_cid(1);

y\_yellow\_cid =-yellow\_cid(2);

y\_cid = -props(cur\_object).Centroid(2);

x\_cid = props(cur\_object).Centroid(1);

k1 = (y\_yellow\_cid- y\_cid)/(x\_yellow\_cid-x\_cid);

y\_s = round(k1\* props(path).BoundingBox(1));

intercept = y\_yellow\_cid - k1\*x\_yellow\_cid;

%% search for next objects

% Arrow direction detection

direction = 1;

if x\_yellow\_cid < x\_cid

direction =-1

else

direction = 1

end

step\_c = 0.01;

found\_point =zeros(0,2);

Hunting\_start\_c = round(yellow\_cid(1));

Hunting\_end\_c = round(yellow\_cid(1)+ 100\* direction);

Hunting\_start\_r = round(yellow\_cid(2));

Hunting\_end\_r = round(yellow\_cid(2)+ 100\* direction);

step = 2\*direction;

if abs(k1) <3.7

for c = Hunting\_start\_c:direction:Hunting\_end\_c

c1 = c + step;

c2 = c + 2\*step;

r = -round(k1\*c+intercept);

r1 = -round(k1\*(c1)+intercept);

r2 = -round(k1\*(c2)+intercept);

if (im(r,c,1)<=8 && im(r,c,2)<=8 && im(r,c,3)<=8) && (im(r1,c1,1)<=251 && im(r1,c1,2)<=251 && im(r1,c1,3)<=251) && (im(r2,c2,1)>=251 && im(r2,c2,2)>=251 && im(r2,c2,3)>=251)

found\_point =[c2,r2];

break;

end

end

else

direction = -((y\_yellow\_cid-y\_cid)/abs(y\_yellow\_cid-y\_cid));

for r = Hunting\_start\_r:direction:Hunting\_end\_r

r1 = r + step;

r2 = r + 2\*step;

c = round((-r-intercept)/k1);

c1 = round((-r1-intercept)/k1);

c2 = round((-r2-intercept)/k1);

if (im(r,c,1)<=8 && im(r,c,2)<=8 && im(r,c,3)<=8) && (im(r1,c1,1)<=251 && im(r1,c1,2)<=251 && im(r1,c1,3)<=251) && (im(r2,c2,1)>=251 && im(r2,c2,2)>=251 && im(r2,c2,3)>=251)

found\_point =[c2,r2];

break;

end

end

end

figure,

dataset\_found\_point = zeros(0,2);

dataset\_found\_point = [dataset\_found\_point;[x\_cid,y\_cid]];

dataset\_found\_point = [dataset\_found\_point;[x\_yellow\_cid,y\_yellow\_cid]];

plot(dataset\_found\_point(:,1),dataset\_found\_point(:,2),'bx')

axis([0 640 -480 0]);

%% find the index

for i = 1: length(Bounding\_box\_stru)

if ismember(found\_point(1),Bounding\_box\_stru(i).Matrix(:,1)) && ismember(found\_point(2),Bounding\_box\_stru(i).Matrix(:,2))

checkpint = 8;

if Bounding\_box\_stru(i).Is\_arrow ==1

cur\_object =Bounding\_box\_stru(i).Index;

assignin('base', 'cur\_object', Bounding\_box\_stru(i).Index);

break;

end

end

end

checkpint = 8;

end

### Lab4 Frame\_difference.m

clear all;

close all;

clc;

%% Parameter declearation

Fontsize = 16;

%%

% read the video

source = VideoReader('car-tracking.mp4');

% create and open the object to write the results

output = VideoWriter('frame\_difference\_output.mp4', 'MPEG-4');

open(output);

thresh = 100; % A parameter to vary

% read the first frame of the video as a background model

bg = readFrame(source);

bg\_bw = rgb2gray(bg); % convert background to greyscale

bg\_bwsize = size(bg\_bw);

% --------------------- process frames -----------------------------------

% loop all the frames

iteration =0;

figure(1) = figure('Color',[1 1 1]);

while hasFrame(source)

fr = readFrame(source); % read in frame

fr\_bw = rgb2gray(fr); % convert frame to grayscale

fr\_diff = abs(double(fr\_bw) - double(bg\_bw)); % cast operands as double to avoid negative overflow

% if fr\_diff > thresh pixel in foreground

fg = uint8(zeros(size(bg\_bw)));

fg(fr\_diff > thresh) = 300;

% update the background model

bg\_bw = fr\_bw;

% visualise the results

figure(1);

subplot(3,1,1), imshow(fr)

title('Video Source input')

subplot(3,1,2), imshow(fr\_bw)

title('Convert Background to Greayscale')

subplot(3,1,3), imshow(fg)

title(['Background subtraction with ';'Frame differencing Detection Threshold =100']);

drawnow

writeVideo(output, fg); % save frame into the output video

iteration = iteration +1;

end

close(output); % save video

Lab4 Gaussian\_mixture\_models.m

close all

clear all

clc ;

% read the video

source = VideoReader('car-tracking.mp4');

% create and open the object to write the results

output = VideoWriter('gmm\_output.mp4', 'MPEG-4');

open(output);

% video frame reading

vidReader = source;

frame\_number=floor(vidReader.Duration \* vidReader.FrameRate);

% create foreground detector object

n\_frames = 500; % a parameter to vary

n\_gaussians = 3; % a parameter to vary

detector = vision.ForegroundDetector('NumTrainingFrames', n\_frames, 'NumGaussians', n\_gaussians);

frame = source.hasFrame

% --------------------- process frames -----------------------------------

% loop all the frames

figure(1) = figure('Color',[1 1 1]);

iteration = 0;

while hasFrame(source)

fr = readFrame(source); % read in frame

fgMask = step(detector, fr); % compute the foreground mask by Gaussian mixture models

% create frame with foreground detection

fg = uint8(zeros(size(fr, 1), size(fr, 2)));

fg(fgMask) = 255;

% visualise the results

figure(1),

subplot(2,1,1), imshow(fr)

title('Video Source Frame')

subplot(2,1,2), imshow(fg)

title('Gaussian mixture models by using Training Frames = 70 and Number of Guansians =4 ')

drawnow

writeVideo(output, fg); % save frame into the output video

iteration = iteration +1 ;

end

close(output); % save video

Lab5 part1\_corner

%% Lab5\_frame reader

close all;

clear all;

clc ;

%% Parameter

Fontsize = 16;

%% Video creator

VidReader = VideoReader('red\_square\_video.mp4');

frame\_number=round(VidReader.Duration \* VidReader.FrameRate);

vidWidth = VidReader.Width;

vidHeight = VidReader.Height;

% Create optical flow object

opticFlow = opticalFlowLK('NoiseThreshold',0.009);

tracks =zeros(0,2);

% Top left point

% Read the first frame

frame\_RGB = readFrame(VidReader);

frame\_Gray = rgb2gray(frame\_RGB);

Corner\_point = corner(frame\_Gray);

corner\_x = min(Corner\_point(:,1));

corner\_y = min(Corner\_point(:,2));

% Add position of this point as the first position in the track;

tracks = [corner\_x,corner\_y];

% initialize the optical flow object

flow = estimateFlow(opticFlow,frame\_Gray);

figure('Color',[1,1,1]);

plot(Corner\_point(:,1),-Corner\_point(:,2),'ro')

axis([0 520 -480 0])

title('Plot conner points in spatial coordinates','Fontsize',Fontsize)

xlabel('x-axi repsent colum of image ')

ylabel('y-axi repsent colum of imag ')

legend('Corner points')

check\_point = 8;

%% figure,

% createfigure1(Corner\_point(:,1),-Corner\_point(:,2));

% axis([0 520 -480 0])

%% Video to frames

mov = struct('cdata',zeros(vidHeight,vidWidth,3,'uint8'),...

'colormap',[]);

% Read one fram at a time until the end of the video is reached

for i = 2:frame\_number

while hasFrame(VidReader)

frame\_RGB = readFrame(VidReader);

frame\_Gray = rgb2gray(frame\_RGB);

Corner\_point = corner(frame\_Gray,'MinimumEigenvalue');

% Find the nearest corner point to your first position from the track

Distance = zeros(0,1);

Matrix\_dis = zeros(0,3);

for i = 1: length(Corner\_point);

Distance = [Distance;sqrt(Corner\_point(i,1) - corner\_x)^2 + (Corner\_point(i,2) - corner\_y)^2];

end

Matrix\_dis = [Corner\_point Distance];

corner\_x = min(Matrix\_dis(:,1));

corner\_y = min(Matrix\_dis(:,2));

% Compute an optical flow for this point (between frames 1 and 2);

flow = estimateFlow(opticFlow,frame\_Gray);

x\_new = corner\_x + flow.Vx(round(corner\_y), round(corner\_x));

y\_new = corner\_y + flow.Vy(round(corner\_y), round(corner\_x));

tracks =[tracks;x\_new,y\_new];

% Visualise the track on the last frame of the video

imshow(frame\_RGB)

hold on

plot(flow,'DecimationFactor',[5 5],'ScaleFactor',10)

title('OpticalFL with Corner Algorithm ','Fontsize',Fontsize)

% axis([0 520 -480 0])

hold off

end

end

%% Plot the results

load red\_square\_gt.mat

figure;

plot(tracks(:,1),tracks(:,2),'r\*');

hold on

plot(gt\_track\_spatial(:,1),gt\_track\_spatial(:,2),'bo');

title(['Squared Errors Ground Truth Values vs estimated track ';'by the optical flow algorithm '],...

'FontSize',Fontsize);

legend('Track by Cornet points','Ground Truth Track','Location','northwest')

figure('Color',[1,1,1]);

for i = 1:length(tracks)

% RMSe(i)= sqrt((tracks(i,1)-gt\_track\_spatial(i,1))^2+(tracks(i,2)-gt\_track\_spatial(i,2))^2)

RMSe(i)= (tracks(i,1)-gt\_track\_spatial(i,1))^2+(tracks(i,2)-gt\_track\_spatial(i,2))^2

bar(i,RMSe(i));

title('Squared Errors','Fontsize',Fontsize)

axis([0 160 0 25])

xlabel('Frames in time serials','Fontsize',Fontsize)

ylabel('square erros','Fontsize',Fontsize)

hold on

end

createfigure9(tracks(:,1),tracks(:,2),gt\_track\_spatial(:,1),gt\_track\_spatial(:,2));

Lab5 Face detection

close all;

clear all;

clc;

%% Parameter

Fontsize = 20;

%% lab5 Section 4

video\_file='visionface.avi';

VidReader=VideoReader(video\_file);

frame\_number=round(VidReader.Duration \* VidReader.FrameRate);

% Create optical flow object

opticFlow = opticalFlowLK('NoiseThreshold',0.009);

%% seprete video to frame

Fra\_index =1:frame\_number;

frame\_RGB = readFrame(VidReader);

frame\_Gray = rgb2gray(frame\_RGB);

%Extract the part of the image

colourRegion = frame\_RGB(132:132 + 80, 279:279+60,:);

imshow(frame\_RGB);

%First constraint extract RGB channels separatelly

red\_channel = colourRegion(:, :, 1);

green\_channel = colourRegion(:, :, 2);

blue\_channel = colourRegion(:, :, 3);

[r\_counts,r\_binLocations] = imhist(red\_channel);

[g\_counts,g\_binLocations] = imhist(green\_channel);

[b\_counts,b\_binLocations] = imhist(blue\_channel);

imhis\_m\_r = [r\_counts,r\_binLocations];

imhis\_m\_g = [g\_counts,g\_binLocations];

imhis\_m\_b = [b\_counts,b\_binLocations];

% nominal r g b

r\_numerator =zeros(0,1);

total\_counter\_r = sum(imhis\_m\_r(:,1));

total\_counter\_g = sum(imhis\_m\_g(:,1));

total\_counter\_b = sum(imhis\_m\_b(:,1));

red\_hist = zeros();

green\_hist = zeros();

blue\_hist = zeros();

for i = 1: length(r\_counts)

red\_hist = [red\_hist;imhis\_m\_r(i,1)/total\_counter\_r];

end

for i = 1: length(g\_counts)

green\_hist = [green\_hist;imhis\_m\_g(i,1)/total\_counter\_g];

end

for i = 1: length(b\_counts)

blue\_hist = [blue\_hist;imhis\_m\_b(i,1)/total\_counter\_b];

end

%%

%Second constraint colour constraint

proximity\_threshold = 3;

%Initialize the optical flow object

flow = estimateFlow(opticFlow,frame\_Gray);

%Read from second frame

for i = 2:frame\_number

while hasFrame(VidReader)

frame\_RGB = readFrame(VidReader);

frame\_Gray = rgb2gray(frame\_RGB);

% imshow(frame\_RGB);

hold on;

bbox = [266, 121, 83, 93];

rectangle('Position', bbox, 'EdgeColor', 'b');

flow = estimateFlow(opticFlow,frame\_Gray);

%Extrac interesting points

[row, col] = find(flow.Vx ~= 0);

interesting\_points = [col, row];

bbox\_metrix = zeros(0,2);

end

end

figure('Color',[1 1 1]),

% plot(interesting\_points(:,1),interesting\_points(:,2),'y.')

% hold on

%bounding box matrix

for i= 266: (266+83)

for j = 121:(121+93)

bbox\_metrix=[bbox\_metrix;[i,j]];

end

end

plot(bbox\_metrix(:,1),bbox\_metrix(:,2),'r.');

hold on

%inteetsting\_points in bounding box

LiA = ismember(interesting\_points,bbox\_metrix,'rows');

interesting\_points\_ind = [interesting\_points LiA];

in\_bbox\_matrix = zeros(0,2);

A=zeros;

%filter points not in bounding\_box

for i = 1:length(interesting\_points)

if interesting\_points\_ind(i,3)== 1

in\_bbox\_matrix = [in\_bbox\_matrix;[interesting\_points\_ind(i,1),interesting\_points\_ind(i,2)]];

A =[A;i];

end

end

%Second constraint

cur\_color = zeros(0,3);

in\_color\_matrix = zeros(0,2);

colour\_threshold = 1.0e-08;

for i = 1:length(in\_bbox\_matrix)

red\_channel = frame\_RGB(in\_bbox\_matrix(i,1), in\_bbox\_matrix(i,2), 1);

green\_channel = frame\_RGB(in\_bbox\_matrix(i,1), in\_bbox\_matrix(i,2), 2);

blue\_channel = frame\_RGB(in\_bbox\_matrix(i,1), in\_bbox\_matrix(i,2), 3);

cur\_color = [red\_channel green\_channel blue\_channel];

if (red\_hist(cur\_color(1) + 1)\* green\_hist(cur\_color(2) + 1) \* blue\_hist(cur\_color(3)+ 1) < colour\_threshold)

in\_color\_matrix = [in\_color\_matrix;[in\_bbox\_matrix(i,1),in\_bbox\_matrix(i,2)]];

else

break;

end

end

x = sum(red\_hist(:,1));

y = sum(green\_hist(:,1));

z = sum(blue\_hist(:,1));

plot(in\_color\_matrix(:,1),in\_color\_matrix(:,2),'bx');

hold on

%%

%% Third constraint proximity constraint of the tracking points

for i = 1: length(in\_color\_matrix)

start\_point = [in\_color\_matrix(i,1),in\_color\_matrix(i,2)];

end

%%

% [s\_row,s\_col] = find(min(in\_color\_matrix(1)));

% start\_point = min(in\_color\_matrix);

start\_point = [in\_color\_matrix(1,1),in\_color\_matrix(1,2)];

% x\_end = max(max(in\_color\_matrix(1)));

x\_end = 317;

% y\_end = max(in\_color\_matrix(2));

y\_end = 214;

D = pdist(in\_color\_matrix,'euclidean');

% for i = 1: length(in\_color\_matrix)

% for j = 1: length(in\_color\_matrix)

% for k =6:100

% in\_prox\_element = [start\_point(1),start\_point(2)+5];

% if (ismember(in\_prox\_element,in\_color\_matrix) )

% in\_prox\_matrix = [in\_prox\_matrix;in\_prox\_element];

% else

% break;

% end

%

% end

% end

% end

%% ¸Ä¶¯µÄµØ·½

startp=1;

in\_prox\_matrix=in\_color\_matrix;

i=0;

while i<length(in\_prox\_matrix)-2

i=i+1;

D=pdist(in\_prox\_matrix,'euclidean');

l=D(startp:startp+length(in\_prox\_matrix)-i-1)<3;

in\_prox\_matrix(logical([zeros(1,i) l]),:)=[];

startp=sum(length(in\_prox\_matrix)-1:-1:length(in\_prox\_matrix)-i)+1;

end

%%

% figure('Color',[1 1 1]);

% plot(bbox\_metrix(:,1),bbox\_metrix(:,2),'r.');

% hold on

%

% plot(in\_color\_matrix(:,1),in\_color\_matrix(:,2),'bx');

% hold on

plot(in\_prox\_matrix(:,1),in\_prox\_matrix(:,2),'gs');

% legend('Points in BoundingBOX','Points meet 2nd constrains','Points meet 3rd Constrains','Fontsize',Fontsize)

title('Interesting Points','Fontsize',Fontsize)

% build cost matrix between current positions and detection points

cost = zeros(size(track\_points, 1), size(detection\_points, 1));

for i = 1:size(track\_points, 1)

for j = 1 : size(detection\_points, 1)

cost(i, j) = norm(track\_points(i, :) - detection\_points(j, :));

end

end

% compute assigments between currents positions and detection points

costOfNonAssignment = 10;

[assignments, unassignedTracks, unassignedDetections] = assignDetectionsToTracks(cost, costOfNonAssignment);