ELEC4630
Assignment 1

Kuang Sheng

45752720

Q1. Street Signs Localization

1.1 Introduction

Digital image processing techniques are widely used in many applications such as street signs localization. This question aims to automatically locate the street signs in the provided images. Both template matching and colour separation techniques were implemented and compared in this specific application.

1.2 Working procedures using template matching

A total number of 11 images containing street signs with different colours and shapes were being evaluated. The first image is illustrated below. As we can see from human eyes, clearly there are four signs in the image.

Figure 1.2.1: Image0

This image contains three signs with diamond shapes and one bigger sign with a rectangle shape below them. The template matching technique was being employed to match the correct street signs. The templates were produced by the software 'Paint' and hand drawing. For example, in this image, the template for the diamond shape was looked like this:

Figure 1.2.2: Original example template

However, the tricky thing is that the three diamond shapes are not identical to the computer because of the shooting angle and space change of the image. Thus, it is difficult to apply a general template for the matching purposes for all diamond shapes signs. Both the image and the template need to be pre-processed.

1.2.1 Image acquisition

Initially, the image and the template were acquired by the computer in digital forms where pixels are the key composition elements of the image.

```
clear, clc, close all;

image = imread('StreetSigns\images0.jpg');
template = imread('StreetSigns\template000.png');
```

1.2.2 Image pre-processing

The image was then pre-processed to remove the noise, enhance, and make the image easier to process. Both the image and the template were converted to a gray scale so that signals we wanted could be enhanced. To make the template more 'general' and remove noise, the template was 'chamfered' by convolving with a Gaussian blurring function. The blurring function smooths images by applying a weighted average of surrounding pixels to each pixel in the image based on Gaussian distribution.

After Gaussian filtering (blurring), the image was converted into binary image based on whether the intensity value was above or below a certain threshold value (150 in this image). A canny edge operator was applied to make the edge clearer eventually. The canny edge operator was widely used in image processing because of its high accuracy. It smooths the images using Gaussian filtering and locates the edges based on Non maximum Suppression and the orientation of gradients. The edges in the image could be detected.

```
%% Image Pre-processing

%Transfer to gray image
gray_img = im2gray(image);
gray_template = im2gray(template);

%Apply blurring function
%gray_img = imgaussfilt(gray_img, 2);
gray_template = imgaussfilt(gray_template, 2);

imshow(gray_template);
%Transfer to black & white image
bw_img = gray_img < 150;


%Apply canny edge operator
edge_template = edge(gray_template, 'canny');
bw_img = edge(bw_img,'canny');
imshow(bw_img);
```

Image after converting into gray scale:

Figure 1.2.3: Image0 in gray scale.

Template after convolving with Gaussian blurring function:

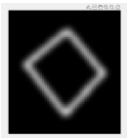


Figure 1.2.4: Example template after blurring.

Image after extracting the edge using Canny edge operator:

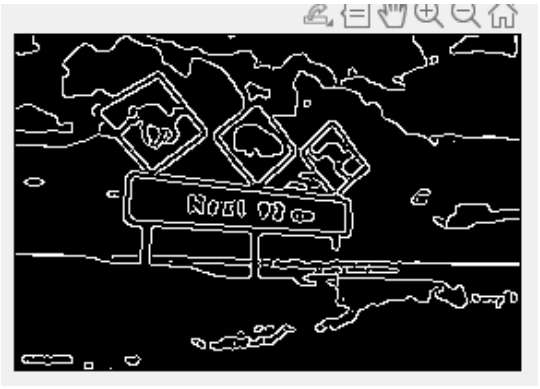


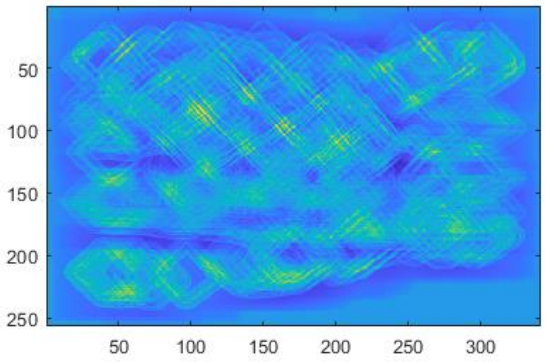
Figure 1.2.5: Image0 using Canny edge operator.

1.2.3 Template Matching

The mathematics behind template matching technique is a 2D cross correlation, where the template refers to the convolution kernel. The 2D cross correlation of the image was accomplished by moving the template over the image. When the centre of the template was aligned with a pixel, each pixel in the surrounding region was multiplied by each weighted coefficient in the template, and the sum of the products was the filtered output of the pixel.

```
25
26 -    %% 2-D cross correleation
27 -    result = normxcorr2(edge_template, bw_img);
28      imagesc(result);
```

The result of the cross correlation was visualized below. The yellow point indicated the locations of high correlations.



To locate only one sign, the location of the highest correlation could be determined.

```
29      %% Locate the road sign
30      % Find the location of the maximum correlation in the result matrix
31 -    [value, index] = max(result(:));
32 -    [max_row, max_col] = ind2sub(size(result), index);
```

Consequently, a rectangular bounding box could be drawn to locate one street sign which matches the template the most.

Apparently, this was not enough as we wanted to locate all four signs at the same time. One of the solutions was nominating a certain threshold for the cross correlation. All the locations above this threshold would be considered as a valid street sign. After testing a variety of threshold values, 0.1494 was the best value in this image with the best localize performance.

```
40      % Find the locations of the matches above the threshold
41 -    threshold = 0.1494;
42 -    [row, col] = find(result >= threshold);
```

In addition, to further extract the information that we wanted, we observed that the street signs have the colours of yellow where the background have other colours. The street signs could therefore be extracted based on the properties of yellow:

```
18 -    rr = image(:,:,1)>200;
19 -    bb = image(:,:,3)<50;
20 -    new_bw = rr & bb;
21 -    imshow(new_bw);
```

The image was easier a lot to process as all the unwanted signals were filtered out. By drawing the rectangular bounding boxes to all the locations where the cross correlation has a value greater than the threshold, the three signs with diamond shapes could be located successfully.

Similar with the above procedures, a blurred template with a rectangle shape could be used to identify and locate the fourth rectangular sign “Next 92km”.

After setting an appropriate threshold value, all four street signs could eventually be located with a relatively poor performance:

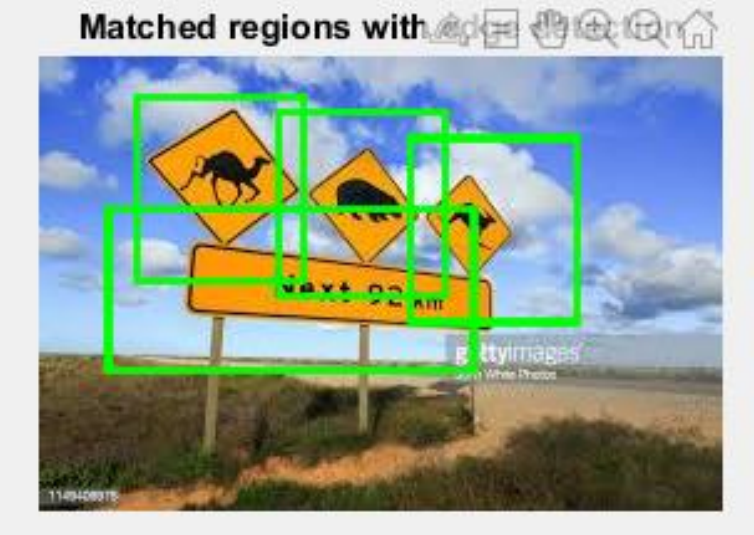


Figure 1.2.6: Street signs localization using template matching in image0

From the above results, template matching seems could be used to locate street signs. More trials have been tested before drawing a conclusion.

Image1

Using other templates such as a combination of diamond and rectangle shapes after blurring:

The same procedure could be used to locate the street sign in ‘image1’:

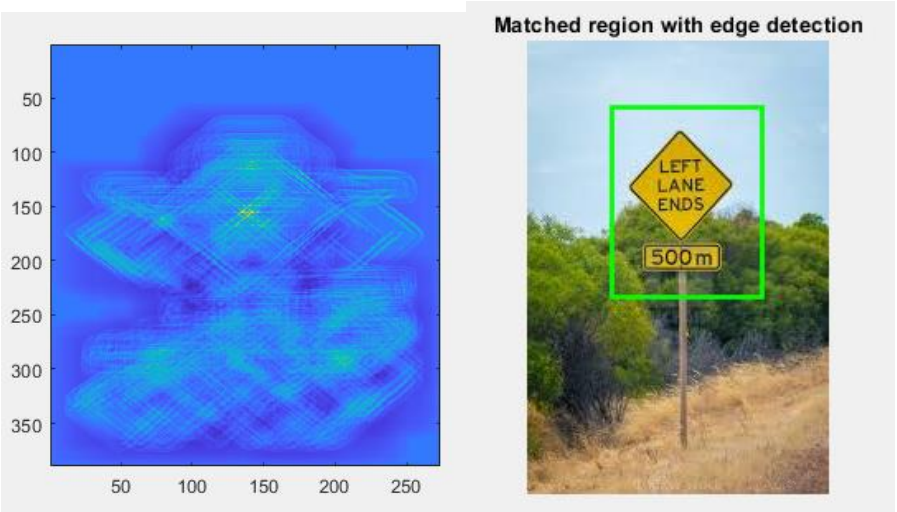
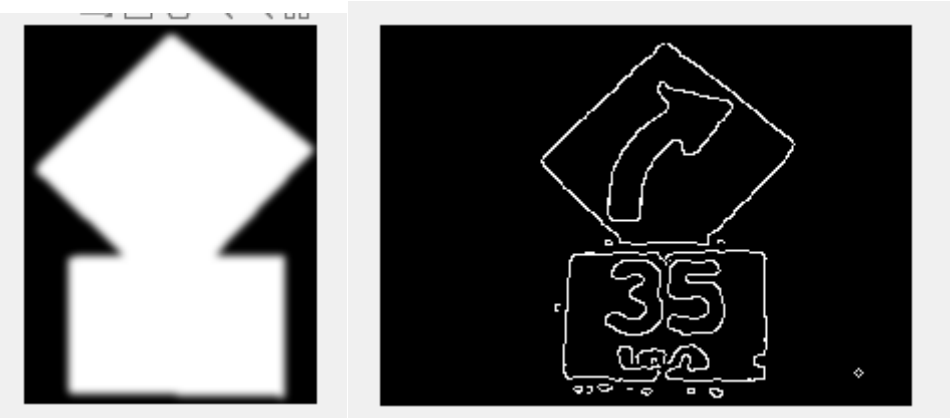


Image2

The problem I met in processing 'image2' was the street sign was too big when did the template matching. By filling the template with a white colour, the feature of the template could be better extracted so that the noise was reduced further:



Therefore, the street sign in image2 could be located after finding the location with highest cross correlation coefficient:

1.2.4 Problems encountered using template matching

After testing a variety of images, template matching technique was found a lot of disadvantages in the application of street signs localization in the deeper processing of other images.

Firstly, it requires a bunch of pre-existing templates. In other words, a template library is needed to match the street signs, it could be computational inefficient as the template library is iterated intend to find an appropriate template, thus more computational resources would be needed resulting in high computational cost and low processing speed.

Secondly, the performance of template matching in applications of street signs localization could be bad because of the uncertainties of shooting angle and space change of the image. In real life situations, the shooting angle of the sign images would not always be identical.

Thirdly, the threshold value for template matching is hard to be determined, every single image would likely to have different threshold values. Consequently, it is hard to develop a 'general' program to process a series of images.

1.3 Working procedures using colour separation.

By observation, most of the street signs have a colour of yellow and red which could be separated from the foreground and background colours. The concepts of dilate and erode in the field of morphology is useful here.

1.3.1 Image pre-processing

Take the first image as example, the four signs with yellow colours could be extracted from the background by adjusting the threshold of RGB. However, due to the presence of the black animal symbols, the initial version of the processed binary image contained contiguous regions where they were expected to be discontiguous. Dilate could be implemented to solve this problem as it made the contiguous regions became discontiguous so that the features of signs could be better extracted. The last thing was removing regions with small areas so that most noise could be removed. The code for pre-processing the images:

```
clear, clc, close all;
%load image
image = imread('StreetSigns\images0.jpg');

%Extracted yellow colour from images
rr = image(:,:,1) > 120;
gg = image(:,:,2) > 100;
bb = image(:,:,3) < 60;
new_bw = rr & gg & bb;
figure, subplot(2,2,1),imshow(image);
subplot(2,2,2),imshow(new_bw);

SE = strel('line', 10, 120);
J = imdilate(new_bw, SE); %Dilate the image using SE
subplot(2,2,3)
imshow(J);
J = imfill(J,'holes');

SE1 = strel('line', 8,0);
J = imerode(J, SE1); %Merode the image using SE1

BW2 = bwareaopen(J,500); %Remove regions less than 500 pixels
BW2 = imerode(BW2, SE1);
subplot(2,2,4)
imshow(BW2);
```

The above code firstly extracts the street signs with yellow colours out from background with different colours. Then it dilates the white colour so that the black components with animal shapes and words would be reduced as the surrounding white components dilated. However, the noise components with white colours were dilated at the same time. To remove the remaining noises, the signs were being eroded and regions less than 500 pixels were forced to remove. The results of pre-processed image were illustrated below:

Figure 1.3.1: Pre-processing image0

1.3.2: Signs localization

After pre-processing, the location of signs could be indicated using a rectangle frame by locating the separate regions:

```
props = regionprops(BW2);

% extract from struct into matrix
rects = cat(1, props.BoundingBox);

hold on;
for i = 1:size(rects,1)
    rectangle('position', rects(i,:), 'EdgeColor', 'g','LineWidth', 3)
end
hold off;
```

As a result, the location of street signs could be indicated with a better performance using colour separation:

Figure 1.3.2: Street signs localization using colour separation.

1.3.3: Outcomes for other images

The same technique could be implemented to most of images.

Outcomes:

As we can see from the above outcomes, most of street signs could now be detected and located correctly using the technique of colour sepration. Note that red colours and yellow colours have different threshold values to extract. In the program I developed, I firstly determined which colour the street sign belongs to, and then extract the features using proposed threshold value. The performance of colour separation was much better than template matching.

1.3.4 Problems encountered using colour separation.

However, there was still a tricky image that was hard to locate the signs using colour, only the sign with a red colour could be detected:

This was because it was hard to find a certain threshold value that has capability to extract red, green, white and black simultaneously. More importantly, the white colour of the street sign is close to the white colour at the background (the sky etc..). Therefore, locating the street signs in this particular image using colour could be a challenge.

1.3.5 Possible solutions

Deep Learning (DL) and Convolutional Neural Networks (CNN) could be implemented to better understand the images and extract features correctly. If lots of information is provided to the system, DL begins to understand it and respond in useful ways. CNN is a sub-network of DL which is commonly used in image processing, it works by learning patterns in image data through layers of convolution and pooling operations. A combination of DL and traditional digital image processing techniques could be effective in real life applications.

Q2. Power Cable Tracking

2.1 Introduction

This question aims to track the intersection of the power cable and the pantograph over time from the provided video. Though both the suspension cable and the power cable are visible, only the power cable is tracked. The procedures for differentiating the two cables have been explained.

2.2 Working Procedures

Since the task was to find the intersection of the power cable and the pantograph, once the location of the pantograph was identical, the coordinates of the intersection could be found using Hough transform. The specific steps were taken as below.

2.2.1 Load frames

Video is essentially made up of images frame by frame, and the provided pantograph video contains 1498 frames in total. Therefore, to locate the power cable in every single frame, the video was split into 1498 images and save to a directory for conveniency.

```
vidObj = VideoReader('Panto2023.mp4');
```

```
%% Load frames to dir 'frames'
if ~exist('./frames','dir')
    mkdir("./frames");
    for image = 1:vidObj.NumFrames  %For each single frame in the video
        filename = strcat('./frames/frame', num2str(image), '.jpg');
        frame = readFrame(vidObj);
        imwrite(frame, filename);  %write current frame to filename
    end
end


% Indicate how many number of framse in the video
% vidObj.NumFrames
```

In addition, since every time looping through the entire frames took a long time, thus only a smaller segment of video was cut out for debugging purposes. Lastly the entire frames would be processed and regenerated a new video.

```
%% Analyze the frames
init_f = 1;  %Initial frame
last_f = 100;  %Last frame

vidframes = read(vidObj,[init_f last_f]);  %Read current frame

video = VideoWriter('./video.mp4', 'MPEG-4');
open(video);

for i = init_f:last_f

    current_f = i + 1 - init_f;  %Current frame
```

2.2.2 Localization of pantograph using Template Matching

In the beginning, the frame image was binarized to extract more features. Take frame 100 as an example:

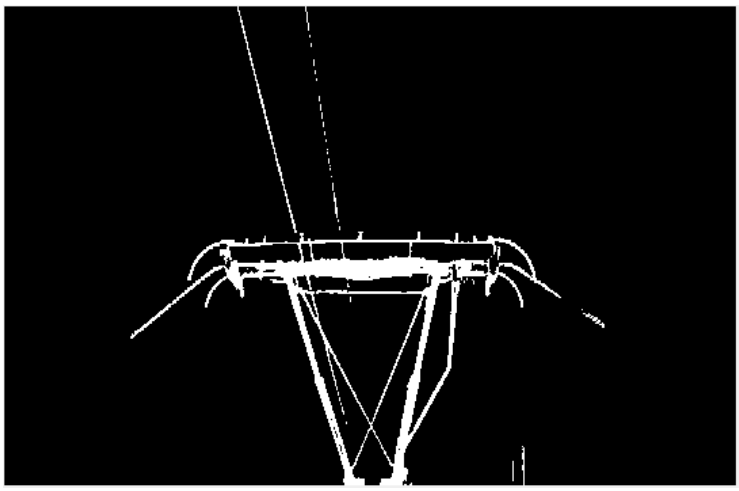


Figure 2.1: Example frame of binarized image

The next step was to generate an appropriate template for matching the pantograph. To save the template locally, either take a screenshot or use image crop in MATLAB is fine.

```
%% Template generate
%template = imcrop(image_bw, [141.5 160.5 257 62]);
template = imread('template1.png');
temp = template(:,:,1) > 200;
```

After cropping from the binarized frame image and removing some noise manually using 'Paint', the template for matching the pantograph was look like this:

Figure 2.2: A general template for matching all frames.

Therefore, the template matching technique could be implemented to locate the pantograph in each frame using a general template picture:

```
    result = normxcorr2(im2double(temp), im2double(image_bw));
%     surf(result); shading flat;

    %Determine the coordinates of the object
    [value, index] = max(result(:));

    [max_row, max_col] = ind2sub(size(result), index);
    template_height = size(template, 1);
    template_width = size(template, 2);
    bbox_x = max_row - template_height + 1;
    bbox_y = max_col - template_width + 1;
```

```
image = insertShape(image, 'rectangle', [bbox_y, bbox_x, template_width, template_height], 'Color', "g", 'LineWidth', 2);
```

The performance for locating the pantograph was pretty good as all frames performed well:

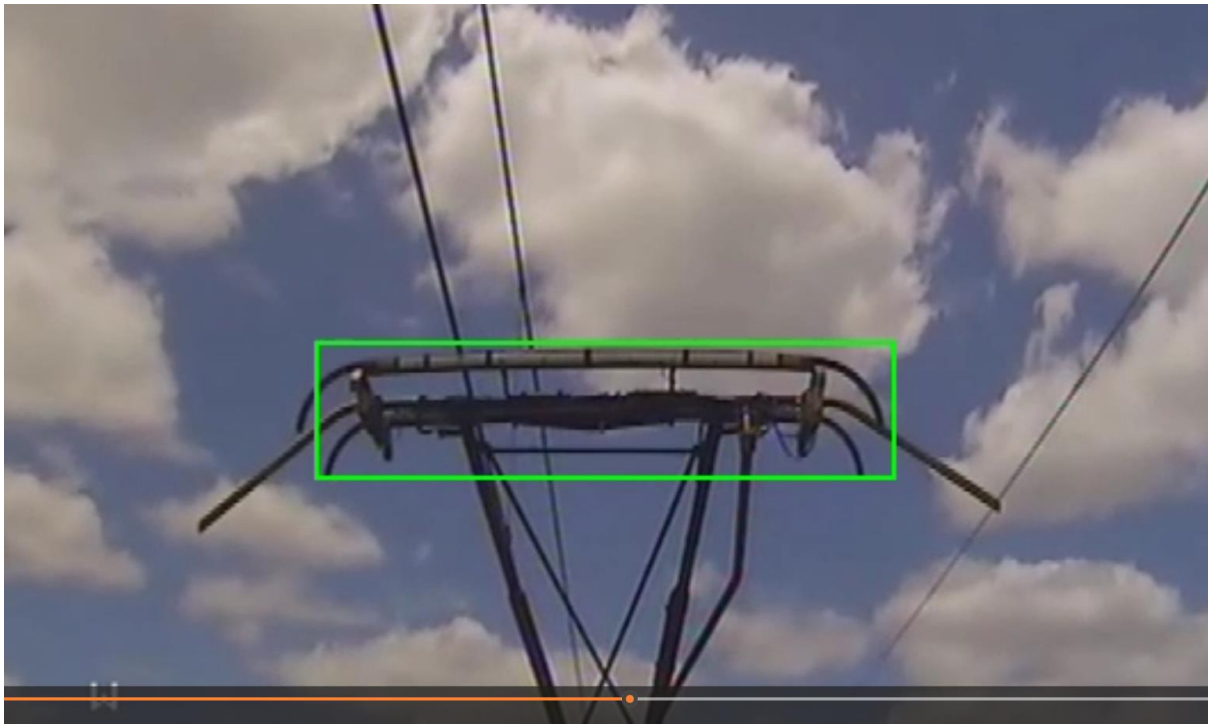


Figure 2.3: Example frame for localization of pantograph using template matching

2.2.3 Localization of the power cable using Hough Transform

Initially, Canny edge operator was applied to make the edges easier to detect.

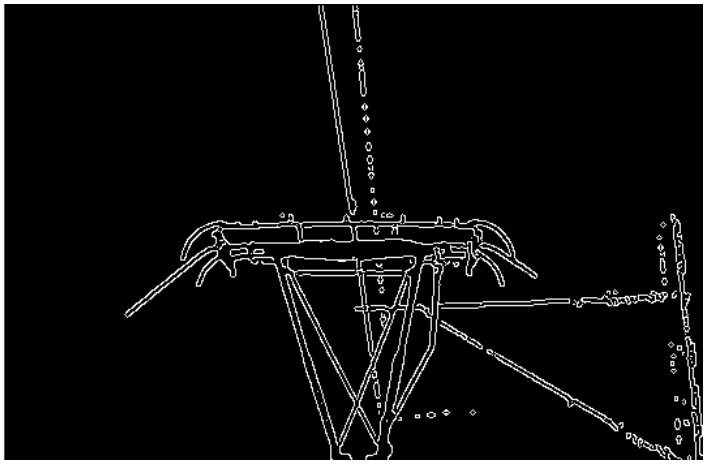
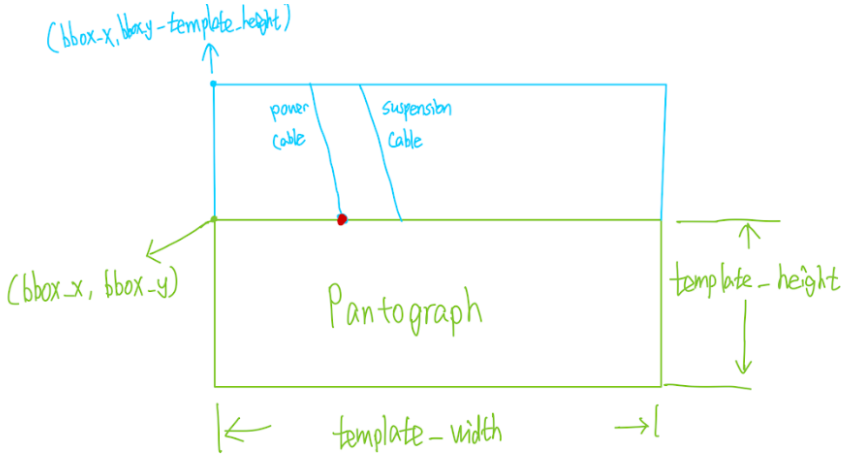


Figure 2.4: Example frame of image using Canny

Refer to Figure 2.3, now that a bounding box has been drawn to locate the pantograph, another bounding box could be cropped right above the pantograph which contains the two cables for most of the frames like this:



Since the coordinates of the pantograph's location were identified, the coordinates of the above blue bounding box could be calculated by shifting coordinates. Therefore, once the relative coordinates of the intersection were produced by Hough transform, the actual coordinates of that intersection in the original frame could be calculated as well. The primary advantage of cropping an additional bounding box was reducing the noises as many as possible so that only two cables were being further processed for most of frames. An example of the additional bounding box:

Figure 2.5: Example frame of two cables

Hough transforms each pixel from the image to a line in a parameter space by representing the slope and y-intercept so that the algorithm is widely used to find simple shapes based on the features. Standard line Hough transform could be used to locate the power cable by finding the longer line in Figure 2.5:

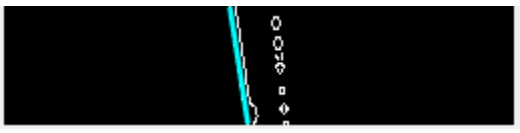


Figure 2.6: Example frame of locating the power cable.

Figure 2.7: Example frame for locating the power cable back project.

To differentiate the power cable and the suspension cable, the longer line was identified as the power cable by determining the endpoints of the longest line segment.

```
%Crop the image containing two lines only
im_line = imcrop(image_canny, [bbox_y bbox_x-template_height template_width template_height]);
figure, imshow(im_line), hold on;
%im_line = bwpropfilt(im_line, 'Area', 1);
[H,T,R] = hough(im_line);

P  = houghpeaks(H,5,'threshold',ceil(0.3*max(H(:))));
x = T(P(:,2)); y = R(P(:,1));

lines = houghlines(im_line,T,R,P,'FillGap',5,'MinLength',7);
%figure, imshow(im_line), hold on;

max_len = 0;
for k = 1:length(lines)
   xy = [lines(k).point1; lines(k).point2];

   % Determine the endpoints of the longest line segment
   len = norm(lines(k).point1 - lines(k).point2);
   if ( len > max_len)
      max_len = len;
      xy_long = xy;  %Find the longer line
   end
end

plot(xy_long(:,1),xy_long(:,2),'LineWidth',2,'Color','cyan');
```

However, though finding the longer line segment could be a way to differentiate two cables, it performed not well for some frames such as frame 200, the suspension cable was accidently mismatched for the power cable:

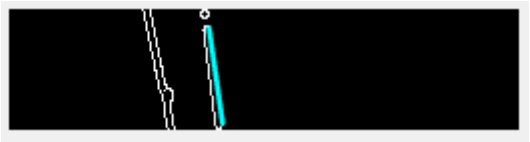


Figure 2.8: Example frame for mismatching

Figure 2.9: Back project to the original frame

To resolve such problem and better differentiate the two cables, the thinner suspension cable could be filtered out by setting an appropriate threshold and the Canny edge operator was abandoned. The result would be like:

Figure 2.10: Remove the suspension cable in binary image

Clearly there is only one cable left for further processing. Hough transform was then applied to locate this cable:

Figure 2.11: Locate the power cable using Hough transform

```
%% Apply Hough Transform to find the cable
[H,T,R] = hough(im_line);

P  = houghpeaks(H,5,'threshold',ceil(0.3*max(H(:))));
x = T(P(:,2)); y = R(P(:,1));

lines = houghlines(im_line,T,R,P,'FillGap',5,'MinLength',7);

max_len = 0;
for k = 1:length(lines)
   xy = [lines(k).point1; lines(k).point2];

   % Determine the endpoints of the longest line segment
   len = norm(lines(k).point1 - lines(k).point2);
   if ( len > max_len)
      max_len = len;
      xy_long = xy;  %Find the longer line
   end
end
```

The last step was recalculating and converting the relative coordinates of the intersection from the cropped image back to the original frame image and indicating that position by drawing a circle:

```
image2 = insertShape(image, 'circle', [xy_long(2,1)+bbox_y xy_long(2,2)+bbox_x-template_height 5],'Color', "r", 'LineWidth', 2);
```

Now that the power cable could be detected with a high accuracy and robust after testing all the frames. To clearly demonstrate the processed video, one frame was taken from every 100 frames to show the outcome:

```
if (mod(current_f,100)==0)
    subplot(5,3,current_f/100);
    imshow(image2);
    title(sprintf('Frame %d', current_f));

end
```

The general outcomes:

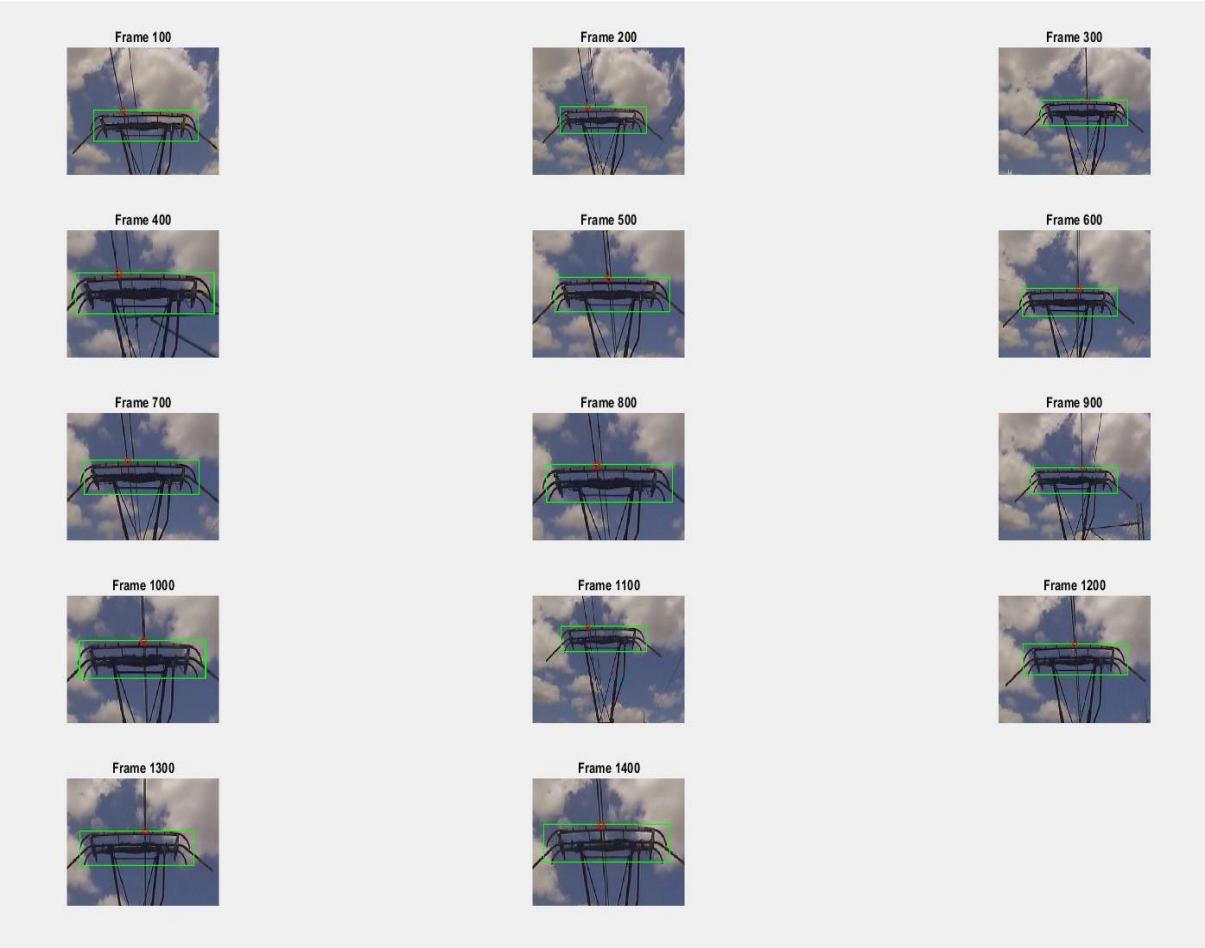


Figure 2.12: Outcomes extracted from the video

Special frames for two or more power cables:

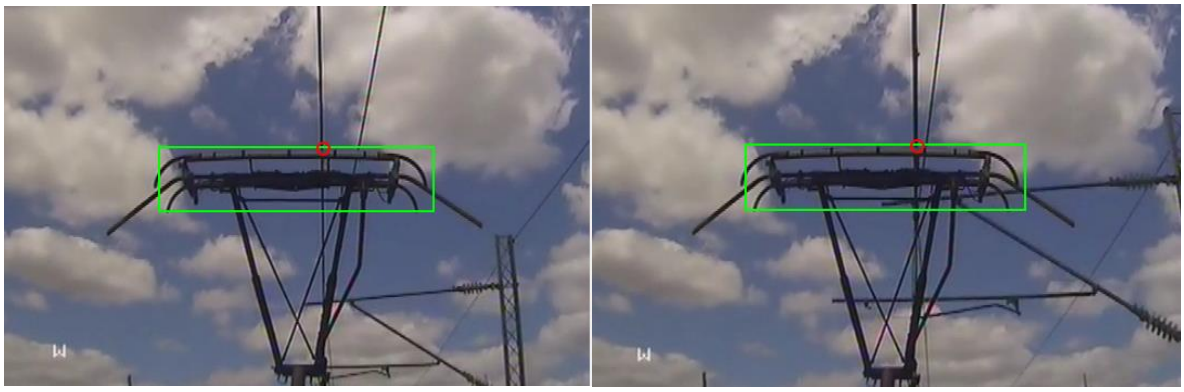


Figure 2.13: Only the current power cable was tracked for frames with more than one power cables

(Example frame 900 and 910)

Apparently, the power cable was located successfully for most of the frames. Therefore, the proposed technique performed well.

2.2.3 Challenges encountered.

However, there were still several frames mismatched mainly concentrated in every time the pantograph passing through the overhead catenary wire. For example,

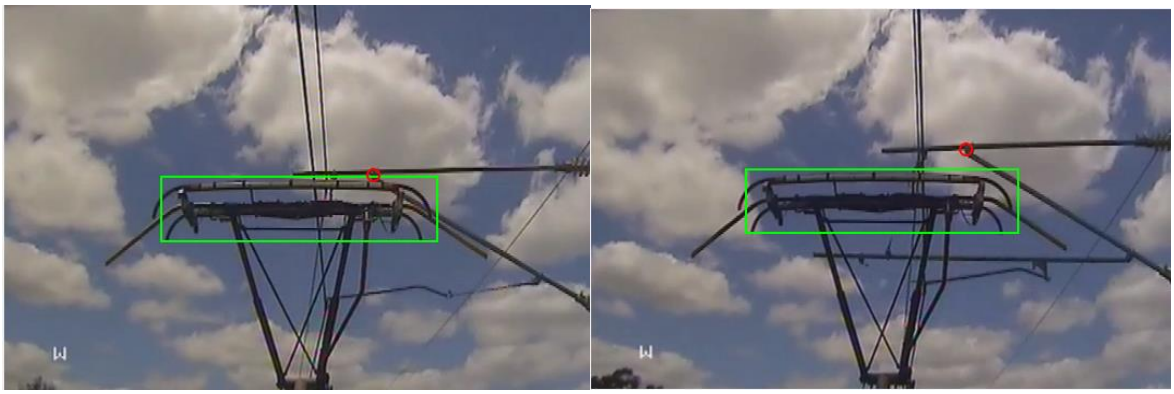


Figure 2.14: Example failed frame 136 and 253

The overhead catenary wires could be a big interference source as it contained longer lines which would be detected by Hough transform. One possible solution was specifying an angle interval so that only vertical lines could be detected while horizontal lines such as the catenary cable would not be considered. After setting up the angle interval:

```
%% Apply Hough Transform to find the cable
[H,T,R] = hough(im_line,'Theta', 1: 0.01: 20);
```

The frame failed previously:

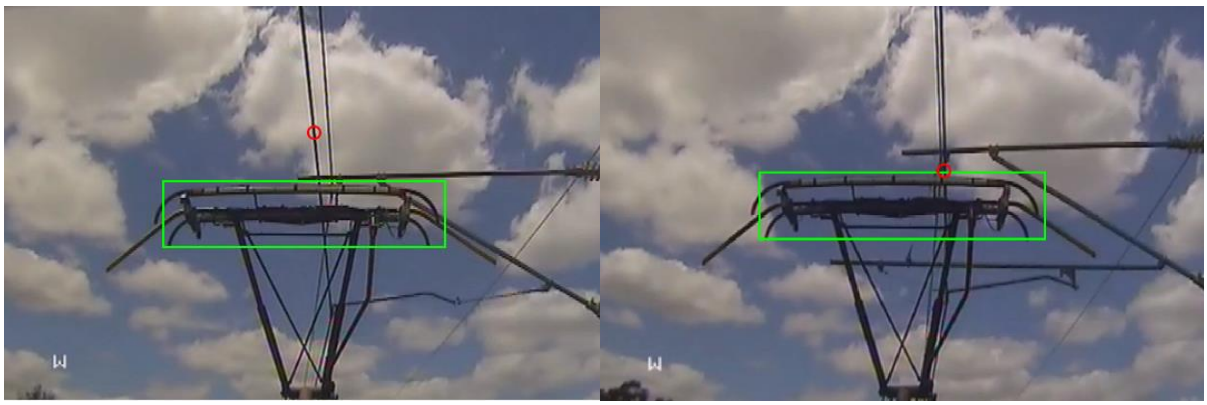


Figure 2.15: Example previous failed frame 136 and 253

Comparing Figure 2.14 and 2.15, as a result, the catenary cable would no longer be mis detected by Hough transform. It seems could be an effective way to resolve this problem.

However, locating the intersection by setting up a fixed angle interval would likely to cause another problem: The intersection point would keep drifting just like what frame 136 illustrated. This probably because only the line segment which most satisfied the proposed features would be indicated. Therefore, it was hard to constantly keep the red circle indicating the correct intersection position. One possible solution would be that fine tuning the parameters for the extracted features such as the angle interval, working in conjunction with algorithms that force to find the intersection position exactly.

Q3. Mr Bean Question

3.1 Introduction

This question intends to find the followings on the image:

- The edge of the road
- Mr Bean's broomstick
- The centres of 2 wheels on the Mini

Both the edge of the road and broomstick have a feature of line shape while the centres of wheels have a feature of circular shape. The key concept being used to solve this question is Hough transformation.

3.2 Methodology

3.2.1 Edge operator selection

After converting image into gray scale, an edge operator could be implemented to detect the edges in the image so that the feature of straight lines could be easier to extracted. The selection of edge operators would depend on specific applications. In this context, Canny and Sobel were considered to detect the edge of the straight lines as both of them are widely used in digital image processing.

Canny is a multi-stage detector involves pre-processing, gradient calculation, suppression, and thresholding. Since it's a relatively sophisticated algorithm, it is more accurate and robust comparing with Sobel operator in many applications, with a tradeoff complex computation consequently. However, due to the canny operator is too accurate in this application, most irrelevant edges at the background such as grass was being detected as demonstrated below:

Figure 3.1: Edge detection using Canny.

Since only the edges of the road and Mr Bean's broomstick were supposed to be detected, apparently canny was not a good choice in this application.

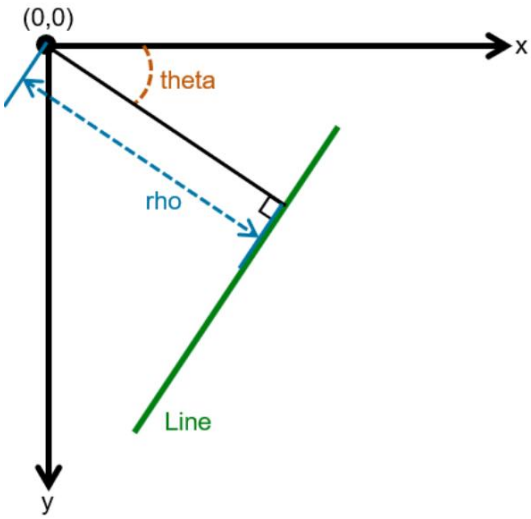
By contrast, Sobel operator is relatively a simple but faster algorithm with lower accuracy. It works by convolving the target image with a small kernel to compute an approximation of the gradient of the image intensity function. In this application, only edges with strong straight line features would be detected successfully. The result was demonstrated below:

Figure 3.2: Edge detection using Sobel.

Clearly Sobel was preferred rather than Canny in this particular application as it reduced most of the noise come from the background.

3.2.2 Line Hough Transformation

After pre-processing the image, standard line hough transformation could be applied to locate the edges of the road and the broomstick which have features of lines. The algorithm transformed each pixel from the image to a line in a parameter space by representing the slope and y-intercept. By observation, the edges of the road could be detected by setting appropriate detection angle so that edges beyond the angle range would not be detected out:




```
%% Standard Line Hough Transform to find road edges
[H,T,R] = hough(im, 'Theta', 75: 0.01: 89);
imshow(H,[],'XData',T,'YData',R,...
            'InitialMagnification','fit');
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, hold on;

P  = houghpeaks(H,1,'threshold',ceil(0.3*max(H(:))));
x = T(P(:,2)); y = R(P(:,1));
plot(x,y,'s','color','white');

lines = houghlines(im,T,R,P,'FillGap',8,'MinLength',180);
figure, imshow(image), hold on
```

In this case the angle range (theta) was set as $75° - 89°$ so that the road edges which almost horizontal could be detected by Hough transform. After plotting the lines as indicators:

```
for k = 1:length(lines)
   xy = [lines(k).point1; lines(k).point2];
   plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','r');

   % Plot beginnings and ends of lines
   plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
   plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','red');

end
```

Figure 3.3: Road edges detection using Hough transform

Therefore, the edges of the road could be located successfully.

Similarly, to detect the broomstick, change the angle range to $10° - 16°$:

```
[H,T,R] = hough(im, 'Theta', 10: 0.01: 16);
P  = houghpeaks(H,1,'threshold',ceil(0.3*max(H(:))));
x = T(P(:,2)); y = R(P(:,1));
plot(x,y,'s','color','white');

lines = houghlines(im,T,R,P,'FillGap',15 ,'MinLength',180);
```

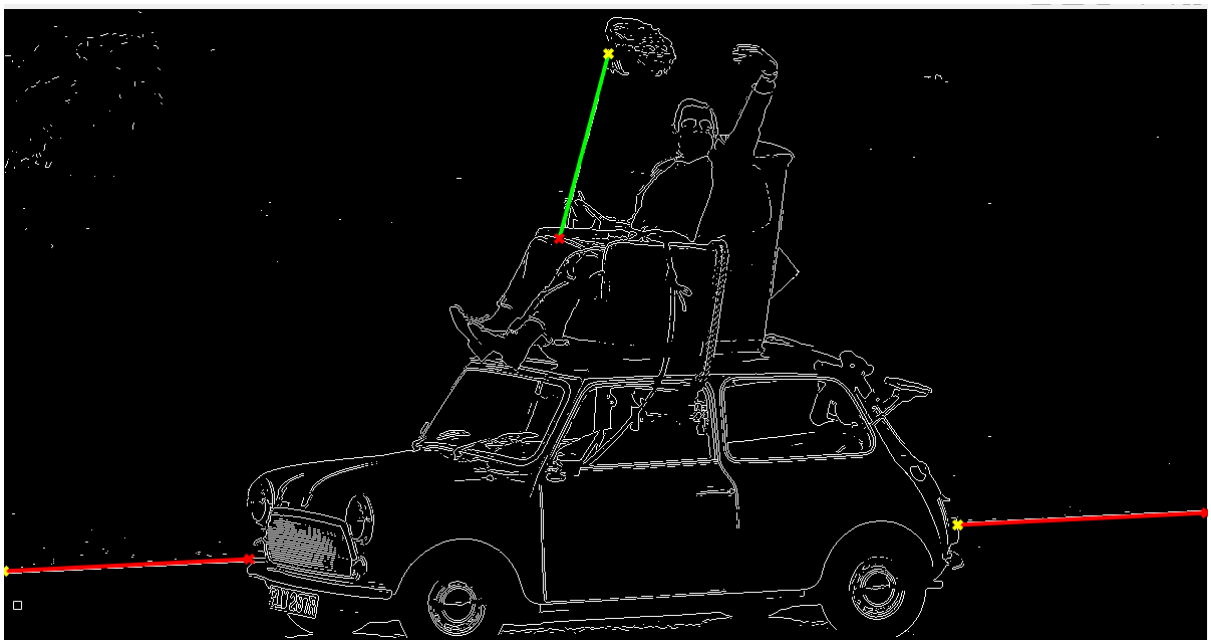


Figure 3.4: Road and broomstick edge detection using Hough transform

Therefore, the broomstick with a feature of vertical straight line could be located successfully.

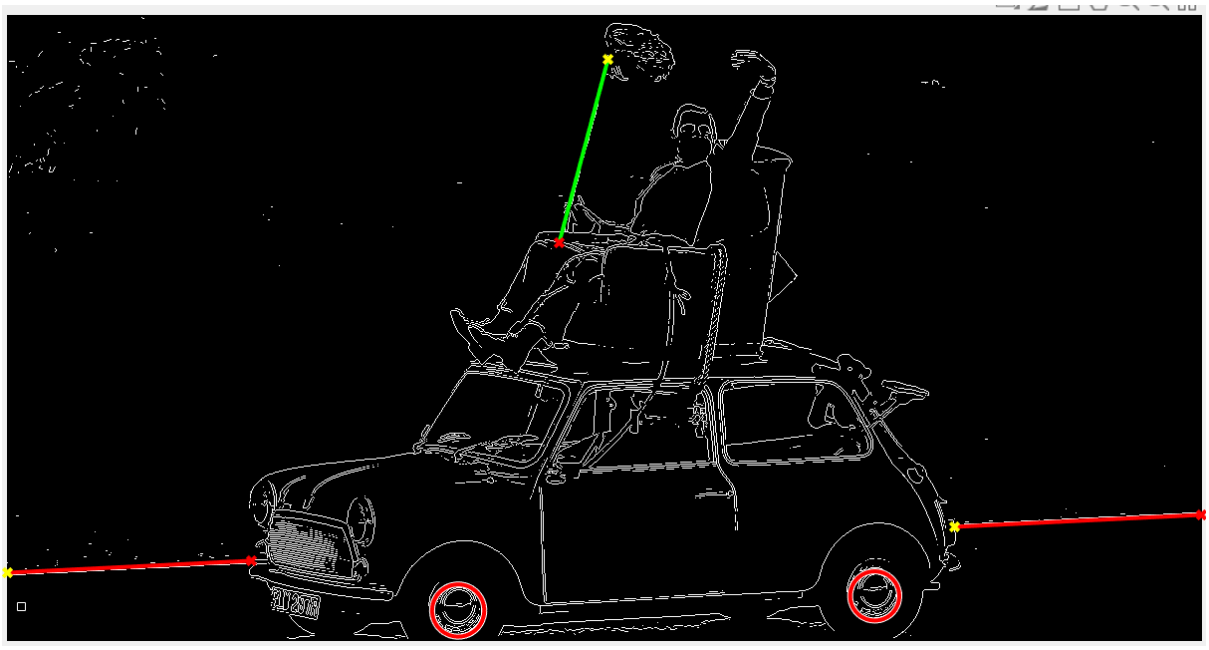
3.2.3 Circular Hough Transformation

Since Hough transform is a powerful feature extraction algorithm, it is not limited to line detections but can be used to detect many simple shapes. The circular Hough transform is used to detect circles in the image by setting circular parameters.

The detection circle radius ranges from 20 to 35 so that the centres of 2 wheels on the Mini could be detected using circular Hough transform:

```
%% Circular Hough Transformation
[centers,radii] = imfindcircles(image,[20,35]);
centersStrong2 = centers(1:2,:);
radiiStrong2 = radii(1:2);

viscircles(centersStrong2, radiiStrong2,'EdgeColor','r');
```



Back project to the original image:

Figure 3.5: Final detection result

Appendix

Code for Q1 using template matching:

```
clear, clc, close all;

image = imread('StreetSigns\images0.jpg');
template = imread('StreetSigns\template000.png');
template1 = imread('StreetSigns\template001.png');
%% Image Pre-processing

%Transfer to gray image
gray_img = im2gray(image);
gray_template = im2gray(template);
gray_template1 = im2gray(template1);

%Apply blurring function
%gray_img = imgaussfilt(gray_img, 2);
gray_template = imgaussfilt(gray_template, 2);
gray_template1 = imgaussfilt(gray_template1, 2);
imshow(gray_template);

rr = image(:,:,1)>200; %135
%gg = image(:,:,2)>100;
bb = image(:,:,3)<50;
new_bw = rr & bb;
imshow(new_bw);

%imshow(gray_template);
%Transfer to black & white image
%new_bw = gray_img < 118;


%Apply canny edge operator
edge_template = edge(gray_template, 'canny');
edge_template1 = edge(gray_template1, 'canny');
bw_img = edge(new_bw,'canny');
imshow(new_bw);


%% 2-D cross corrleation
result = normxcorr2(edge_template, bw_img);
result1 = normxcorr2(edge_template1, bw_img);
%imagesc(result);

% Find the locations of the matches above the threshold
threshold = 0.1494;
threshold1 = 0.2;
[row, col] = find(result >= threshold);
[row1, col1] = find(result1 >= threshold1);

% Obtain the coordinates of the bounding boxes
template_height = size(template, 1);
template_width = size(template, 2);
bbox_top = row - template_height + 1;
bbox_left = col - template_width + 1;
```

```
bbox_bottom = row;
bbox_right = col;

% Obtain the coordinates of the bounding boxes
template_height1 = size(template1, 1);
template_width1 = size(template1, 2);
bbox_top1 = row1 - template_height1 -10;
bbox_left1 = col1 - template_width1;
bbox_bottom1 = row1;
bbox_right1 = col1;

% Draw rectangle to locate the signs
figure;
imshow(image);
hold on;
for j = 1:length(row)
    rectangle('Position', [bbox_left(j), bbox_top(j),
template_width, template_height], 'EdgeColor', 'g', 'LineWidth', 2);
end
for i = 1:length(row1)
    rectangle('Position', [bbox_left1(i), bbox_top1(i),
template_width1, template_height1], 'EdgeColor', 'g', 'LineWidth',
2);
end
title('Matched regions with edge detection');
```

Code for Q1 using colour separation (preferred technique):

```
clear, clc, close all;

image = imread('StreetSigns\images9.jpg');

rr = image(:,:,1) > 135; %135
bb = image(:,:,2) < 80;
new_bw = rr & bb;
imshow(new_bw);


SE = strel('disk', 2);
J = imdilate(new_bw, SE);
%imshow(J);

J = imfill(J,'holes');

BW2 = bwareaopen(J,700);
figure, imshow(BW2);
imshow(image);


props = regionprops(BW2);

% extract from struct into matrix
rects = cat(1, props.BoundingBox);
yell = 0;
yblue = 0;
hold on;
for i = 1:size(rects,1)
    rectangle('position', rects(i,:), 'EdgeColor', 'g','LineWidth',
3)
    yell = 1; %set the flag
    yblue = 1;
end
hold off;

if yell == 0
    rr = image(:,:,1) > 120; %135
    gg = image(:,:,2) > 100;
    bb = image(:,:,3) < 60;
    new_bw = rr & gg & bb;
    imshow(new_bw);

    SE = strel('line', 10, 120);
    J = imdilate(new_bw, SE); %peng zhang
    imshow(J);
    J = imfill(J,'holes');

    SE1 = strel('line', 8,0);
    J = imerode(J, SE1);

    BW2 = bwareaopen(J,600);
    BW2 = imerode(BW2, SE1);
    figure, imshow(BW2);
```

```
    imshow(image);

    props = regionprops(BW2);

    % extract from struct into matrix
    rects = cat(1, props.BoundingBox);

    hold on;
    for i = 1:size(rects,1)
        rectangle('position', rects(i,:), 'EdgeColor',
'g','LineWidth', 3)
        yblue = 1;
    end
    hold off;


end

if yblue == 0
    rr = image(:,:,1) > 135; %135
bb = image(:,:,2) < 180;
new_bw = rr & bb;

imshow(new_bw);

SE = strel('disk', 2);
J = imdilate(new_bw, SE);
imshow(J);

J = imfill(J,'holes');

BW2 = bwareaopen(J,700);
imshow(image);


props = regionprops(BW2);

% extract from struct into matrix
rects = cat(1, props.BoundingBox);

hold on;
for i = 1:size(rects,1)
    rectangle('position', rects(i,:), 'EdgeColor', 'g','LineWidth',
3)
end
hold off;
end
```

Code for Q2:

```
clear, clc, close all;
matlab.video.read.UseHardwareAcceleration('off');

vidObj = VideoReader('Panto2023.mp4');

%% Load frames to dir 'frames'
if ~exist('./frames','dir')
    mkdir("./frames");
    for image = 1:vidObj.NumFrames  %For each single frame in the
video
        filename = strcat('./frames/frame', num2str(image), '.jpg');
        frame = readFrame(vidObj);
        imwrite(frame, filename);  %write current frame to filename
    end
end


% Indicate how many number of framse in the video
% vidObj.NumFrames

%% Analyze the frames
init_f = 1;  %Initial frame
last_f = 1498;  %Last frame

vidframes = read(vidObj,[init_f last_f]);  %Read current frame

video = VideoWriter('./video.mp4', 'MPEG-4');
open(video);

for i = init_f:last_f

    current_f = i + 1 - init_f;  %Current frame

    %Original image crop
    orig_im = vidframes(:,:,:,current_f);
    image = imcrop(orig_im,[147.5 42.5 539 352]);
    %imshow(image);

    gray = rgb2gray(image);  %Convert to gray scale
    image_bw = gray(:,:,1) < 50;  %Binarize
    image_canny = edge(image_bw,'canny');
    %figure,imshow(image_canny);
    %imshow(image_canny);


    %template = imcrop(image_bw, [141.5 160.5 257 62]);
    template = imread('template1.png');
    %[a,b] = size(template);
    temp = template(:,:,1) > 200;

    result = normxcorr2(im2double(temp), im2double(image_bw));
%     surf(result); shading flat;

    %Determine the coordinates of the object
```

```
    [value, index] = max(result(:));

    [max_row, max_col] = ind2sub(size(result), index);
    template_height = size(template, 1);
    template_width = size(template, 2);
    bbox_x = max_row - template_height + 1;
    bbox_y = max_col - template_width + 1;

    %Crop the image containing two lines only
    im_line = imcrop(image_bw, [bbox_y bbox_x-template_height
template_width template_height]);
    im_line = bwpropfilt(im_line, 'Area', 1);
    %figure, imshow(im_line), hold on;
    %% Apply Hough Transform to find the cable
    [H,T,R] = hough(im_line);

    P  = houghpeaks(H,5,'threshold',ceil(0.3*max(H(:))));
    x = T(P(:,2)); y = R(P(:,1));

    lines = houghlines(im_line,T,R,P,'FillGap',5,'MinLength',7);

    max_len = 0;
    for k = 1:length(lines)
       xy = [lines(k).point1; lines(k).point2];

       % Determine the endpoints of the longest line segment
       len = norm(lines(k).point1 - lines(k).point2);
       if ( len > max_len)
          max_len = len;
          xy_long = xy;  %Find the longer line
       end
    end

    %plot(xy_long(:,1),xy_long(:,2),'LineWidth',2,'Color','cyan');


    image = insertShape(image, 'rectangle', [bbox_y, bbox_x,
template_width, template_height], 'Color', "g", 'LineWidth', 2);
    image2 = insertShape(image, 'circle', [xy_long(2,1)+bbox_y
xy_long(2,2)+bbox_x-template_height 5],'Color', "r", 'LineWidth',
2);

%     if (mod(current_f,100)==0)
%         subplot(5,3,current_f/100);
%         imshow(image2);
%         title(sprintf('Frame %d', current_f));
%
%     end
    %imshow(image2);


    writeVideo(video, image2);


end
close(video);
```

Code for Q3 using Hough transform:

```
%% Pre-processing data
image = imread('MrBean2023.jpg');
im = im2gray(image);
im = edge(im, 'Sobel');
imshow(im);

%% Standard Line Hough Transform to find road edges
[H,T,R] = hough(im, 'Theta', 75: 0.01: 89);
imshow(H,[],'XData',T,'YData',R,...
            'InitialMagnification','fit');
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, hold on;

P  = houghpeaks(H,1,'threshold',ceil(0.3*max(H(:))));
x = T(P(:,2)); y = R(P(:,1));
plot(x,y,'s','color','white');

lines = houghlines(im,T,R,P,'FillGap',8,'MinLength',180);
figure, imshow(image), hold on

for k = 1:length(lines)
   xy = [lines(k).point1; lines(k).point2];
   plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','r');

   % Plot beginnings and ends of lines
   plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
   plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','red');

end

[H,T,R] = hough(im, 'Theta', 10: 0.01: 16);
P  = houghpeaks(H,1,'threshold',ceil(0.3*max(H(:))));
x = T(P(:,2)); y = R(P(:,1));
plot(x,y,'s','color','white');

lines = houghlines(im,T,R,P,'FillGap',15 ,'MinLength',180);

max_len = 0;
for k = 1:length(lines)
   xy = [lines(k).point1; lines(k).point2];
   plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');

   % Plot beginnings and ends of lines
   plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
   plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','red');

end

%% Circular Hough Transformation
[centers,radii] = imfindcircles(image,[20,35]);
centersStrong2 = centers(1:2,:);
radiiStrong2 = radii(1:2);

viscircles(centersStrong2, radiiStrong2,'EdgeColor','r');
```