

Importing and opening an image

OpenCV (Open Computer source vision) is a library of programming mainly aimed at real-time computer vision. In this case cv2 is used to read and show the image.

```
1 import cv2
2
3 image = cv2.imread('picture.jpg')
4 cv2.imshow('result', image)
5 cv2.waitKey(0)
6
```

Changing the image to grayscale

We first make a copy of the original image using numpy (np) so as to have a copy of the original picture at the end. After renaming the copy, we can then make editions on it.

```
4 image = cv2.imread('picture.jpg')
5 lane_image = np.copy(image)
6 gray = cv2.cvtColor(lane_image, cv2.COLOR_RGB2GRAY)
```

Smoothing

Blurring the picture into a 5 x 5 matrix box; this is done so as to remove the sharp edges in the picture and leave nice curves so as to increase the accuracy. The Gaussian blur function smoothens the picture.

```
7 blur = cv2.GaussianBlur(gray, (5, 5), 0)
8 cv2.imshow('result', blur)
```

Canny image

The canny function detects the sharp changes in pixel values, we usually use a ratio of 1:3, in this case using a lower and higher threshold of 50 and 150 respectively.

```
7 blur = cv2.GaussianBlur(gray, (5, 5), 0)
8 canny = cv2.Canny(blur, 50, 150)
```

The result is as follows:



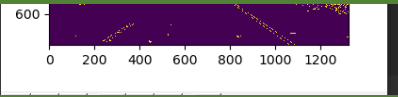
Drawing the picture on an axis

We will then use pyplot to print the picture with an axis so as to see the values we can use when cropping the image to select the section that we need; the section that has road lanes.

```

16
17 plt.imshow(canny)
18 plt.show()
19

```



Region of interest

The image shape gives an array (rows, columns), so by taking `shape[0]`, we get the height of the image. The image is then selected as a polygon. A mask image is formed with all pixel numbers at 0 (black), then filled with the polygon image at 255 (white). This produces is white triangle on a black background.

```

def region_of_interest(image):
    height = image.shape[0]
    polygon = np.array([
        [(200, height), (1100, height), (550, 250)]
    ])
    mask = np.zeros_like(image)
    cv2.fillPoly(mask, polygon, 255)
    return mask

```

Cropping the image

We use the `bitwise_and` function to link our image (canny image) with the mask. The rest of the image remains black since any pixel compared with 0 by 'and' remains 0. The white lines however, remain white as the 1 and 1 gives 1.

```

masked_image = cv2.bitwise_and(image, mask)
return masked_image

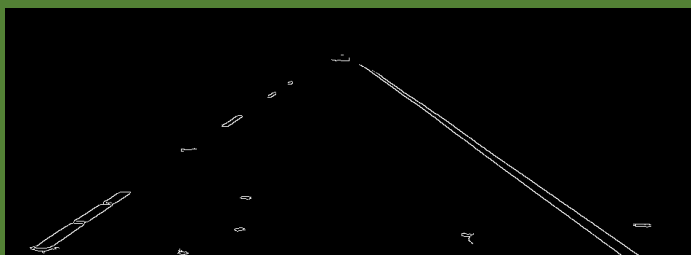
```

```

cropped_image = region_of_interest(canny)
cv2.imshow('result', cropped_image)
cv2.waitKey(0)

```

And the result is as below



Detecting lines

Hough is a method used to detect lines in an image. From the cropped image, we use 2 pixels for a bin and 1r (angle), we use 100 votes per bin, to be the minimum number of votes required for a bin to be chosen as a point of intersection. Next parameter is a replacement array (empty array).

```

lines = cv2.HoughLinesP(cropped_image, 2, np.pi/180, 100, np.array([]), minLineLength=40, maxLineGap=5)
line_image = display_lines(lane_image, lines)

```

Drawing lines function

Takes 2 parameters, lines detected and the image. First create a black image, same size as the original image. The lines object is a list with 4 items, x1, y1, x2, y2. We then use these points to create lines using a cv2 line function, set the coordinates for every loop and set the color to blue, last parameter is the thickness of the lines to be drawn. Lastly, we return the line image and call the function to detect and draw the lines of a black image.

```
def display_lines(image, lines):  
    line_image = np.zeros_like(image)  
    if lines is not None:  
        for line in lines:  
            x1, y1, x2, y2 = line.reshape(4)  
            cv2.line(line_image, (x1, y1), (x2, y2), (255, 0, 0), 10)  
    return line_image
```

Add-weighted

In this part we merge the lane image (original) with the line image (detected lines) to become one pic. The lane image is multiplied by a factor of 0.8, and line image by 1, making it 20% more intense than lane image.

```
combo = cv2.addWeighted(lane_image, 0.8, line_image, 1, 1)
```

Average slope intercept

Function with image and lines, first we create empty arrays to add in the parameters of the averaged lines. We then get all the points for each line by looping, then use function polyfit which will return a list with the slope and the y-intercept for the points given. We then check to see if the slope is (+) or (-) so as to determine whether it's the left or right line, and finally append the parameters into the respective variables. Next step; find the average of the lines, to give one slope value and one intercept value for each line.

```
def average_slope_intercept(image, lines):  
    left_fit = []  
    right_fit = []  
    for line in lines:  
        x1, y1, x2, y2 = line.reshape(4)  
        parameters = np.polyfit((x1, x2), (y1, y2), 1)  
        intercept = parameters[1]  
        slope = parameters[0]  
        if slope < 0:  
            left_fit.append((slope, intercept))  
        else:  
            right_fit.append((slope, intercept))  
    left_fit_average = np.average(left_fit, axis=0)  
    right_fit_average = np.average(right_fit, axis=0)
```

Make coordinates

Take out the slope and intercept from the array, then calculate the coordinates of the 4 points. Point y_2 is $3/5$ away from 0, y_1 is the y_1 value since `image.shape[0]` returns the number of row (pixels) in the picture. We then use the line equation to find the values of x_1 and x_2 .

```
def make_coordinates(image, line_parameters):
    slope, intercept = line_parameters
    y1 = image.shape[0]
    y2 = int(y1 * (3/5))
    x1 = int((y1 - intercept)/slope)
    x2 = int((y2 - intercept)/slope)
    return np.array([x1, y1, x2, y2])
```

We then apply the `make coordinates` function to the image, using the averaged lines.

```
left_line = make_coordinates(image, left_fit_average)
right_line = make_coordinates(image, right_fit_average)
```

Playing a video

We start by calling the video, then read it to get (a boolean, a frame) for each slide, wait key, release and destroy all windows functions are used to close the window when done.

```
cap = cv2.VideoCapture('test2.mp4')
while cap.isOpened():
    _, frame = cap.read()
    canny_image = canny(frame)
    cropped_image = region_of_interest(canny_image)
    lines = cv2.HoughLinesP(cropped_image, 2, np.pi / 180, 100, np.array([]), minLineLength=40, maxLineGap=5)
    averaged_lines = average_slope_intercept(frame, lines)
    line_image = display_lines(frame, averaged_lines)
    combo = cv2.addWeighted(frame, 0.8, line_image, 1, 1)
    cv2.imshow('media', combo)
    if cv2.waitKey(1) == ord('s'):
        break
cap.release()
cv2.destroyAllWindows()
```

