**自動駕駛實務**

**Project\_1**

**Lane Line Detection**

**電機所 碩一**

**廖聲宇**

**N26121143**

**1.Import Library**

1. import matplotlib.pyplot as plt
2. import matplotlib.image as mpimg
3. import numpy as np
4. import cv2
5. import math
6. from PIL import Image
7. from moviepy.editor import VideoFileClip

引入需要用到的函示庫

**2.Define image process function and show image**

1. def pipeline(image):
2. ysize = image.shape[0]
3. xsize = image.shape[1]
4. def apply\_smoothing(image, kernel\_size = 3):
5. return cv2.GaussianBlur(image, (kernel\_size, kernel\_size), 0)
6. def select\_white\_yellow(image):
7. converted = convert\_hls(image)
8. # white color mask
9. lower = np.uint8([ 0, 160, 0])
10. upper = np.uint8([255, 255, 255])
11. white\_mask = cv2.inRange(converted, lower, upper)
12. # yellow color mask
13. lower = np.uint8([ 10, 0, 100])
14. upper = np.uint8([ 40, 255, 255])
15. yellow\_mask = cv2.inRange(converted, lower, upper)
16. # combine the mask
17. mask = cv2.bitwise\_or(white\_mask, yellow\_mask)
18. return cv2.bitwise\_and(image, image, mask = mask)
19. picking\_white\_yellow = select\_white\_yellow(image)
21. gaussian\_image = apply\_smoothing(picking\_white\_yellow)
22. # Call Canny Edge Detection here.
23. cannyed\_image = cv2.Canny(gaussian\_image, 50, 220)
24. plt.figure()
25. plt.imshow(cannyed\_image)

首先將需要的影像處理撰寫成function，並建立一個pipeline包含所有function，由第2、3行來獲得影像的尺寸，接著說明影像處理的步驟及函示。

**以下測試圖片為test.jpg/test02.jpg**

|  |  |
| --- | --- |
| test.jpg | test02.jpg |
|  |  |

第5、6行建立一個**高斯模糊的函示**，來調整影像中的噪點。

第8~20行建立一個**擷取出黃色/白色邊線的函示**，其中第11~13行為給定需要擷取的黃色上下限範圍，第15~17行為給定需要擷取的白色上下限範圍，針對只有白線的test.jpg僅設定白色範圍即可，接著由第20行執行將兩個顏色擷取結果合併成一張影像，結果如下圖。

|  |  |
| --- | --- |
| test.jpg | test02.jpg |
|  |  |

將上面的影像由第24行執行**高斯模糊**處理，可以得到下圖。

|  |  |
| --- | --- |
| test.jpg | test02.jpg |
|  |  |

將上面的影像由第26行執行**邊緣偵測**處理找出影像中的輪廓，可以得到下圖。

|  |  |
| --- | --- |
| test.jpg | test02.jpg |
|  |  |

說明完影像處理的步驟及函示，接著說明影像想要**裁切(辨識)的區塊**，如此可以減少道路邊線以外的物體影響。

1. def region\_of\_interest(img, vertices):
2. mask = np.zeros\_like(img)
3. # channel\_count = img.shape[] \* channel\_count/下面的
4. match\_mask\_color = (255,)
5. cv2.fillPoly(mask, vertices, match\_mask\_color)
6. masked\_image = cv2.bitwise\_and(img, mask)
7. return masked\_image
8. #
9. region\_of\_interest\_vertices = [
10. (xsize\*0.16, ysize),
11. (xsize\*0.42, ysize\*0.65),
12. (xsize\*0.52, ysize\*0.65),
13. (xsize\*0.92, ysize),
14. ]
15. cropped\_image = region\_of\_interest(
16. cannyed\_image,
17. np.array([region\_of\_interest\_vertices], np.int32),)
18. plt.figure()
19. plt.imshow(cropped\_image)

第1~20行為定義出想要**裁切的影像區塊**，給定四個頂點，裁出一個梯形區塊，並對上面邊緣偵測完的影像進行裁切，結果如下圖。

實際裁切的區域

|  |  |
| --- | --- |
| test.jpg | test02.jpg |
|  |  |

將邊緣偵測影像進行裁切的結果

|  |  |
| --- | --- |
| test.jpg | test02.jpg |
|  |  |

再來對上面的影像進行**霍夫轉換**處理**，**來找出車道邊線的線段。**霍夫轉換(**Hough Transform)是一種用於檢測幾何形狀的經典圖像處理技術，特別是用於檢測直線和圓形。

1. def draw\_lines(img, lines, color=[255, 0, 0], thickness=6):
2. # If there are no lines to draw, exit.
3. if lines is None:
4. return
5. # Make a copy of the original image.
6. img = np.copy(img)
7. # Create a blank image that matches the original in size.
8. line\_img = np.zeros(
9. (
10. img.shape[0],
11. img.shape[1],
12. 3
13. ),
14. dtype=np.uint8,
15. )
16. # Loop over all lines and draw them on the blank image.
17. for line in lines:
18. for x1, y1, x2, y2 in line:
19. cv2.line(line\_img, (x1, y1), (x2, y2), color, thickness)
20. # Merge the image with the lines onto the original.
21. img = cv2.addWeighted(img, 0.95, line\_img, 1.0, 0.0)
22. # Return the modified image.
23. return img
24. #Hough
25. lines = cv2.HoughLinesP(
26. cropped\_image,
27. rho=2,
28. theta=np.pi / 180,
29. threshold=10,
30. lines=np.array([]),
31. minLineLength=15,
32. maxLineGap=300
33. )
34. line\_image = draw\_lines(image, lines)
35. plt.figure()
36. plt.imshow(line\_image)
37. # plt.show()
38. left\_line\_x = []
39. left\_line\_y = []
40. right\_line\_x = []
41. right\_line\_y = []
42. for line in lines:
43. for x1, y1, x2, y2 in line:
44. slope = (y2 - y1) / (x2 - x1) # <-- Calculating the slope.
45. if math.fabs(slope) < 0.5: # <-- Only consider extreme slope
46. continue
47. if slope <= 0: # <-- If the slope is negative, left group.
48. left\_line\_x.extend([x1, x2])
49. left\_line\_y.extend([y1, y2])
50. else: # <-- Otherwise, right group.
51. right\_line\_x.extend([x1, x2])
52. right\_line\_y.extend([y1, y2])
53. min\_y = int(image.shape[0] \* 3.25 / 5 )# <-- Just below the horizon
54. max\_y = image.shape[0] # <-- The bottom of the image
55. poly\_left = np.poly1d(np.polyfit(
56. left\_line\_y,
57. left\_line\_x,
58. deg=1
59. ))
60. left\_x\_start = int(poly\_left(max\_y))
61. left\_x\_end = int(poly\_left(min\_y))
62. poly\_right = np.poly1d(np.polyfit(
63. right\_line\_y,
64. right\_line\_x,
65. deg=1
66. ))
67. right\_x\_start = int(poly\_right(max\_y))
68. right\_x\_end = int(poly\_right(min\_y))
69. line1\_image = draw\_lines(image,[[
70. [left\_x\_start, max\_y, left\_x\_end, min\_y],
71. [right\_x\_start, max\_y, right\_x\_end, min\_y]]], thickness=6,)
72. plt.figure()
73. plt.imshow(draw\_lines\_image)
74. plt.show()
75. return draw\_lines\_image

第1~78行為在**影像畫出車道線**。

第3~23行，首先建立複製影像img，透過霍夫轉換找出裁切影像上直線的集合，並在line\_img上繪出檢測到的線條，最後將line\_img與img疊合，即可得到繪有線條的影像，如下圖。

霍夫轉換後的結果

|  |  |
| --- | --- |
| test.jpg | test02.jpg |
|  |  |

第26~34行為**霍夫轉換的參數設置**，其中:

**rho** -累加器的距離解析度，以像素為單位。

**theta** -累加器的角度解析度，以弧度為單位。

**threshold** -累加器的閾值參數。只有獲得足夠投票的直線（> 閾值）才會被返回。

**minLineLength** -最小線段長度。 小於此長度的線段將被拒絕。

**maxLineGap** -同一直線上兩點之間允許的最大間隙。

第40~73行為**擬合出車道線**，首先會檢測霍夫轉換得到的直線，對於每條直線先計算其斜率（slope），並檢查斜率的絕對值是否大於某個閾值（0.5），排除過於平緩的直線，只保留較陡的車道線。 根據斜率的正負將直線劃分為左右兩組，將各組的端點座標（x1，y1）和（x2，y2）新增到對應的清單中。 然後將左右兩組座標點進行多項式擬合，以估計車道線的位置，這裡使用了np.polyfit 函數進行一次多項式擬合。擬合後得到車道線在圖像中的起始點和結束點，使用擬合後的多項式函數來計算車道線在影像底部和視野最遠處的位置。

最後對上面的影像**擬合出視野中的車道線**

|  |  |
| --- | --- |
| test.jpg | test02.jpg |
|  |  |

**讀取影像及影片**的部分

1. #圖片
2. white\_output = 'test\_output.jpg'
3. clip1 = VideoFileClip("test.jpg")
4. #white\_output = 'test02\_output.jpg'
5. #clip1 = VideoFileClip("test02.jpg")
6. #黃線
7. # white\_output = 'solidYellowLeft\_output.mp4'
8. # clip1 = VideoFileClip("solidYellowLeft.mp4")
9. #白線
10. # white\_output = 'solidWhiteRight\_output.mp4'
11. # clip1 = VideoFileClip('solidWhiteRight.mp4')
12. #挑戰
13. # white\_output = 'challenge\_output.mp4'
14. # clip1 = VideoFileClip("challenge.mp4")
15. #國道
16. # white\_output = '/guodao\_output.mp4'
17. # clip1 = VideoFileClip("/guodao.mp4")
18. white\_clip = clip1.fl\_image(pipeline)
19. white\_clip.write\_videofile(white\_output, audio=False)

第2~5行為讀取影像的部分。

第7~17行為讀取影片的部分。

第19、20行為對影片中的每一幀進行pipeline函示的處理，並輸出成影片。

最終結果:

|  |  |
| --- | --- |
| 影片辨識結果 | |
| 白線 | [自駕車實務\_solidWhiteRight output (youtube.com)](https://www.youtube.com/watch?v=9W6dw1b68LM) |
| 黃線 | [自駕車實務\_solidYellowLeft output (youtube.com)](https://www.youtube.com/watch?v=rxAWoGkKAyA) |
| 挑戰 | [自駕車實務\_challenge output (youtube.com)](https://www.youtube.com/watch?v=Y4m36cE0YsM) |
| 國道 | [自駕車實務\_guodao output (youtube.com)](https://www.youtube.com/watch?v=vSzCJhe9i3s) |