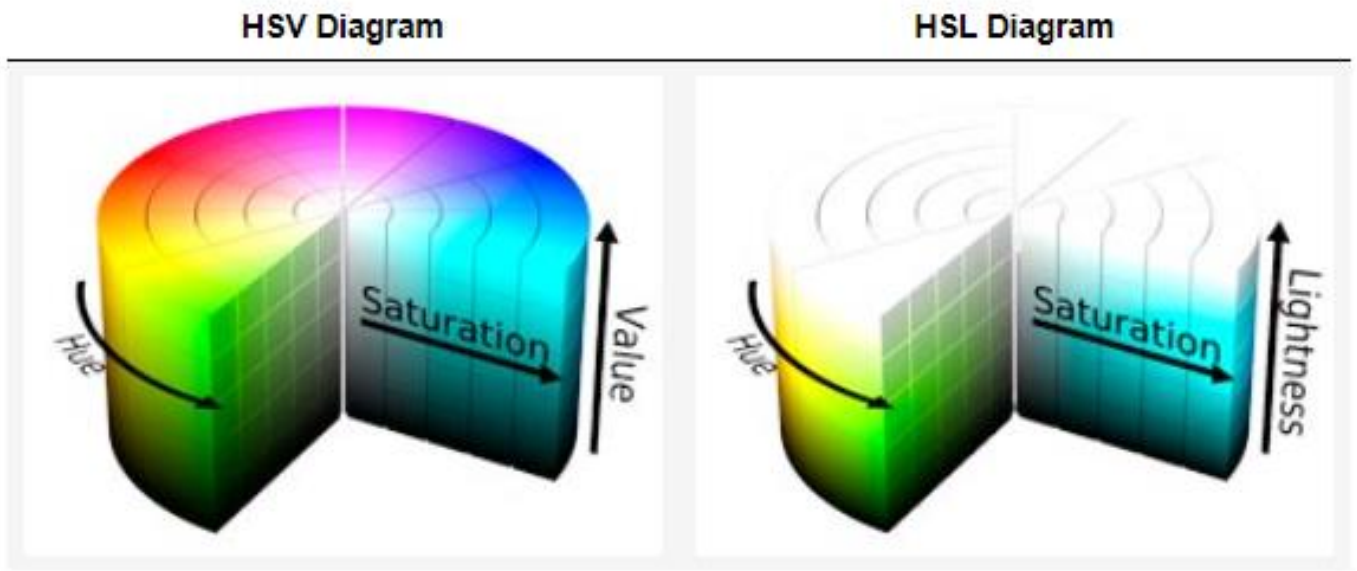
自動駕駛實務: Project1

學號:N26091819
姓名:卓奕全
日期: 4/21

把原圖轉換為hsl圖像

```
def to_hsv(img):
    return cv2.cvtColor(img, cv2.COLOR_RGB2HSV)

def to_hsl(img):
    return cv2.cvtColor(img, cv2.COLOR_RGB2HLS)
```



- 這邊有hsv與hsl的比較圖
- 我們可以看到hsl圖像整體較為明亮
- 因為我們希望可以更明顯辨識白色與黃色的道路線，所以可以發現hsl圖像較適合

分離出黃色與白色

```python
def isolate_yellow_hsl(img):
    # Caution - OpenCV encodes the data in ****HLS*** format
    # Lower value equivalent pure HSL is (30, 45, 15)
    low_threshold = np.array([15, 38, 115], dtype=np.uint8)
    # Higher value equivalent pure HSL is (75, 100, 80)
    high_threshold = np.array([35, 204, 255], dtype=np.uint8)

    yellow_mask = cv2.inRange(img, low_threshold, high_threshold)

    return yellow_mask


# Image should have already been converted to HSL color space
def isolate_white_hsl(img):
    # Caution - OpenCV encodes the data in ***HLS*** format
    # Lower value equivalent pure HSL is (30, 45, 15)
    low_threshold = np.array([0, 200, 0], dtype=np.uint8)
    # Higher value equivalent pure HSL is (360, 100, 100)
    high_threshold = np.array([180, 255, 255], dtype=np.uint8)

    yellow_mask = cv2.inRange(img, low_threshold, high_threshold)

    return yellow_mask
```

- 從HSL圖像中分離出黃色和白色
- 分別設定要黃色跟白色在hsl圖像中的上下限值(low_threshold、high_threshold)
- cv2.inRange():只顯示設定範圍內的顏色

轉為灰度圖

```python
def grayscale(img):
    """Applies the Grayscale transform
    This will return an image with only one color channel
    but NOTE: to see the returned image as grayscale
    (assuming your grayscaled image is called 'gray')
    you should call plt.imshow(gray, cmap='gray')"""
    return cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    # Or use BGR2GRAY if you read an image with cv2.imread()
    # return cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

- 圖像轉成灰階圖來處理，用COLOR_RGB2GRAY
- 我們感興趣的是檢測圖像上的白線或黃線，當圖像為灰度時會顯示出特別高的對比度。
- 把因為道路是黑色的，所以道路上任何較亮的東西都會在灰度圖像中以高對比度顯示出來，如右圖。

高斯模糊

```
def gaussian_blur(img, kernel_size):
    """Applies a Gaussian Noise kernel"""
    return cv2.GaussianBlur(img, (kernel_size, kernel_size), 0)
```

- 高斯模糊（也稱為高斯平滑）是一種用於平滑圖像邊緣以減少noise的預處理技術
- Kernel_size愈大，模糊的程度愈高

Canny 邊緣檢測

```
def canny(img, low_threshold, high_threshold):
    """Applies the Canny transform"""
    return cv2.Canny(img, low_threshold, high_threshold)
```

現在我們已經對圖像進行了充分的預處理，可以應用canny，其作用是識別圖像中的線條並丟棄所有其他數據。

- 第一個參數是需要處理的原圖象，且該圖必須為單通道的灰階圖
- 第二個參數是較小的threshold
- 第三個參數是較大的threshold
- 其中較大的threshold用於檢測圖像中明顯的邊緣，但一般情況下檢測的效果不會那麼完美，邊緣檢測出來是斷斷續續的。所以這時候用較小的第一個threshold將這些間斷的邊緣連接起來。
- 函數最後返回一個二值圖，其中包含檢測出的邊緣

只保留感興趣的區域

```python
def region_of_interest(img, vertices):
    """
    Applies an image mask.

    Only keeps the region of the image defined by the polygon
    formed from `vertices`. The rest of the image is set to black.
    `vertices` should be a numpy array of integer points.
    """
    #defining a blank mask to start with
    mask = np.zeros_like(img)

    #defining a 3 channel or 1 channel color to fill the mask with depending on the input image
    if len(img.shape) > 2:
        channel_count = img.shape[2]  # i.e. 3 or 4 depending on your image
        ignore_mask_color = (255,) * channel_count
    else:
        ignore_mask_color = 255

    #filling pixels inside the polygon defined by "vertices" with the fill color
    cv2.fillPoly(mask, vertices, ignore_mask_color)

    #returning the image only where mask pixels are nonzero
    masked_image = cv2.bitwise_and(img, mask)
    return masked_image
```

- 確定感興趣的區域，並丟棄該多邊形之外的所有線。
- 這邊要特別注意，不同長寬的圖片 要去做不同的設定
- fillpoly(): 選擇區域
- Bitwiae_add(): 在圖像上屏蔽不感興趣的區域

分隔左右車道

```python
def separate_lines(lines, img):
    img_shape = img.shape

    middle_x = img_shape[1] / 2

    left_lane_lines = []
    right_lane_lines = []

    for line in lines:
        for x1, y1, x2, y2 in line:
            dx = x2 - x1
            if dx == 0:
                #Discarding line since we can't gradient is undefined at this dx
                continue
            dy = y2 - y1

            # Similarly, if the y value remains constant as x increases, discard line
            if dy == 0:
                continue

            slope = dy / dx

            # This is pure guess than anything...
            # but get rid of lines with a small slope as they are likely to be horizontal one
            epsilon = 0.1
            if abs(slope) <= epsilon:
                continue

            if slope < 0 and x1 < middle_x and x2 < middle_x:
                # Lane should also be within the left hand side of region of interest
                left_lane_lines.append([[x1, y1, x2, y2]])
            elif x1 >= middle_x and x2 >= middle_x:
                # Lane should also be within the right hand side of region of interest
                right_lane_lines.append([[x1, y1, x2, y2]])

    return left_lane_lines, right_lane_lines
```

Challenge:

- lines是儲存在陣列中偵測到要畫的點
- 左右兩邊的線分開處理
- 左車道：隨著x值（即寬度）增加，y值（即高度）減少：斜率因此必須為負
- 右車道：隨著x值（即寬度）增加，y值（即高度）增加：斜率因此必須為正

劃線

```
def draw_lines(img, lines, color=[255, 0, 0], thickness=8):
    """
    NOTE: this is the function you might want to use as a starting point once y
    average/extrapolate the line segments you detect to map out the full
    extent of the lane (going from the result shown in raw-lines-example.mp4
    to that shown in P1_example.mp4).

    Think about things like separating line segments by their
    slope ((y2-y1)/(x2-x1)) to decide which segments are part of the left
    line vs. the right line.  Then, you can average the position of each of
    the lines and extrapolate to the top and bottom of the lane.

    This function draws `lines` with `color` and `thickness`.
    Lines are drawn on the image inplace (mutates the image).
    If you want to make the lines semi-transparent, think about combining
    this function with the weighted_img() function below
    """
    for line in lines:
        for x1,y1,x2,y2 in line:
            if abs(y1 - y2) > 20 and abs (abs(y1 - y2) / abs(x1 - x2)) >= 0.5:
                cv2.line(img, (x1, y1), (x2, y2), color, thickness)
```

- lines是儲存在陣列中偵測到要畫的點
- color [255,0,0]為劃出來的線顏色，紅色
- thickness調整劃線的粗細
- 另外這邊額外用斜率來做進一步的判斷，因為車道線通常不會太斜，所以這邊去掉太過於斜的線
- 兩個點要劃線而他們的y軸通常會有一定的距離

霍夫轉換

```python
def hough_lines(img, rho, theta, threshold, min_line_len, max_line_gap):
    """
    `img` should be the output of a Canny transform.

    Returns an image with hough lines drawn.
    """
    lines = cv2.HoughLinesP(img, rho, theta, threshold, np.array([]), minLineLength=min_line_len, maxLineGap=max_line_gap)
    line_img = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)
    draw_lines(line_img, lines)
    return line_img
```

霍夫轉換是一種特徵提取方法，用於檢測圖像中的簡單形狀，例如圓形，直線

- rho: 分辨率參數p以像素為單位
- theta: 參數的分辨率θ以弧度為單位
- threshold: 檢測一條線的最小相交點數
- min_line_len: 劃線所需的最小pixels數量
- maxLineGap: 線之間的最大間隔距離，給得越大就會把線給相連起來

Pipeline的流程

Github上3個測試影片的pipeline:

```
def lane_detection_pipeline(self, img):
    combined_hsl_img = filter_img_hsl(img)
    grayscale_img = grayscale(combined_hsl_img)
    gaussian_smoothed_img = gaussian_blur(grayscale_img, kernel_size=5)
    canny_img = canny_edge_detector(gaussian_smoothed_img, 50, 150)
    segmented_img = region_of_interest(canny_img)
    hough_lines = hough_transform(segmented_img, rho, theta, threshold, min_line_length, max_line_gap)
```

- 將原始圖像轉換為HSL
- 從HSL圖像中分離出黃色和白色
- 將圖像轉換為灰度以便於操作
- 應用高斯模糊以平滑邊緣
- 在平滑的灰度圖像上應用Canny Edge Detection
- 跟踪感興趣的區域，並丟棄上一步確定的該區域之外的所有其他行
- 執行霍夫變換以在我們感興趣的區域內找到車道，並用黑色對其進行追踪
- 外推以創建兩條平滑線

Pipeline的流程

加分題影片的pipeline:

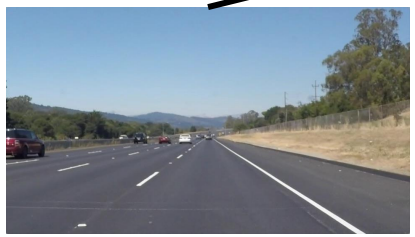
```
def process_image(image):
    # NOTE: The output you return should be a color image (3 channel) for processing video below
    # TODO: put your pipeline here,
    # you should return the final output (image where lines are drawn on lanes)
    gray_image = grayscale(image)
    gaus_blur = gaussian_blur(gray_image, 3)
    edges = canny(gaus_blur, 200,400)
    imshape = image.shape

    ##vertices = np.array([[(0,imshape[0]),(450, 320), (500, 320), (imshape[1],imshape[0])]], dtype=np.int32)
    vertices = np.array([[(100, 560),(400, 420), (560, 420), (900, 560)]], dtype=np.int32)
    masked = region_of_interest(edges, vertices)

    rho = 1             #distance resolution in pixels of the Hough grid
    theta = np.pi/180   #angular resolution in radians of the Hough grid
    threshold = 15      #minimum number of votes (intersections in Hough grid cell)
    min_line_len = 10   #minimum number of pixels making up a line
    max_line_gap = 150   #maximum gap in pixels between connectable line segments
    line_image = hough_lines(masked, rho, theta, threshold, min_line_len, max_line_gap)

    result = weighted_img(line_image, image)
    return result
```

- 首先將圖片轉為灰度圖
- 利用高斯模糊讓線條更明顯，高斯模糊程度為3
- 用canny選擇要保留的線條這邊設(200,400)
- 設定感興趣的區域，也就是只保留左右車道線的範圍，該區域為一個梯形
- 設定霍夫轉換的參數然後呼叫hough_line，來提取影片中的線條
- 最後呼叫weighted_img把已劃線的圖片傳回

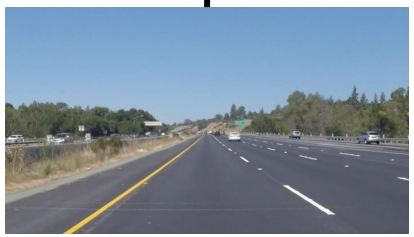
變動調整的參數

Hough lines 中的參數:

```
rho = 1
# 1 degree
theta = (np.pi/180) * 1
threshold = 15
min_line_length = 20
max_line_gap = 10
```



solidWhiteRight



solidYellowLeft

challenge

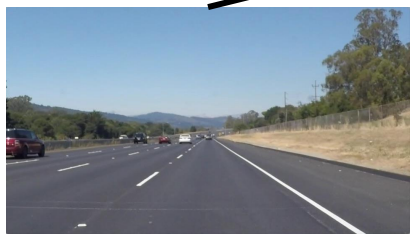
```
rho = 1
theta = np.pi/180
threshold = 15
min_line_len = 10
max_line_gap = 150
```

加分題

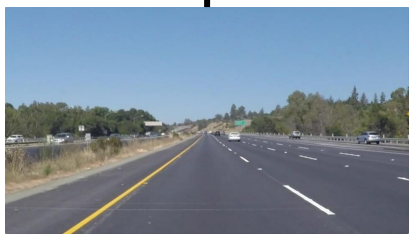
變動調整的參數

Canny 中的參數:

```
canny_img = canny_edge_detector(gaussian_smoothed_img, 50, 150)
```



solidWhiteRight



solidYellowLeft

challenge

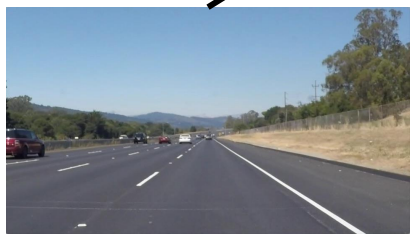
```
edges = canny(gaus_blur, 200,400)
```

加分題

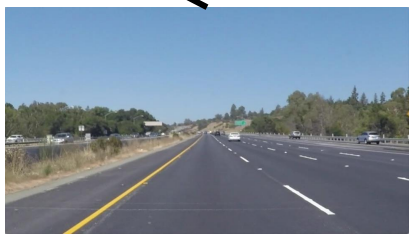
變動調整的參數

Pixel: 960 x540

```
region_bottom_left = (130 ,img_shape[0] - 1)
region_top_left = (410, 330)
region_top_right = (650, 350)
region_bottom_right = (img_shape[1] - 30,img_shape[0] - 1)
```



solidWhiteRight



solidYellowLeft

Pixel: 1280 x 720

```
region_bottom_left = (200 , 680)
region_top_left = (600, 450)
region_top_right = (750, 450)
region_bottom_right = (1100, 650)
```

challenge

```
vertices = np.array([[(100, 560),(400, 420), (560, 420), (900, 560)]], dtype=np.int32)
masked = region_of_interest(edges, vertices)
```

加分題

影片連結

solidWhiteRight

https://youtu.be/JyEpV0byPds

solidYellowLeft

https://youtu.be/DIz6oXPhl48

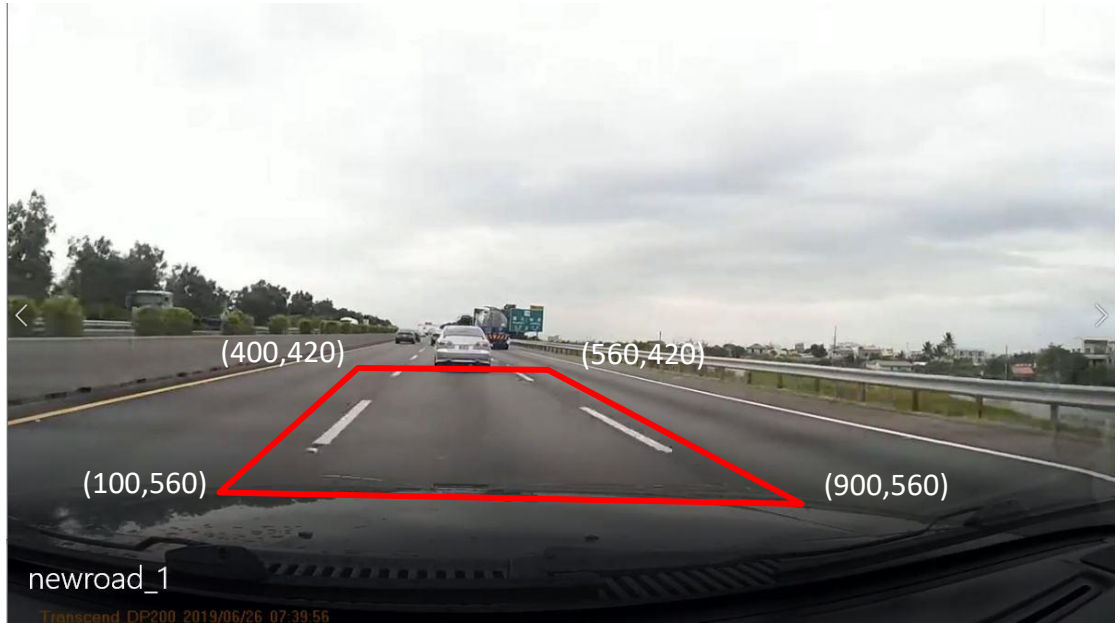
challenge

https://youtu.be/EJmZozWHYBY

加分題影片

(0,0)

(1280,0)



(0,720)

(1280,720)

```
vertices = np.array([[(100, 560),(400, 420), (560, 420), (900, 560)]], dtype=np.int32)
masked = region_of_interest(edges, vertices)
```

我的貢獻:
1.手動調整要保留的劃線區域
2.利用斜率來判斷，刪去一些不必要的線段，只留下最主要　的左右車道線
3.調整每張圖片之間的劃線間格，使他能連成一直線
4調整canny使偵測到的線在理想的範圍內

上傳到youtube的網址: https://youtu.be/8A_TyAo3q-g

原影片 網址: https://www.youtube.com/watch?v=kS9Cm4vXxho

遇到問題參考資料

1. OS error: https://stackoverflow.com/questions/43966523/getting-oserror-winerror-6-the-handle-is-invalid-in-videofileclip-function
2. Process_image (pipeline)參考: https://ithelp.ithome.com.tw/articles/10203636?sc=pt
3. drawlines function參考: https://github.com/udacity/CarND-LaneLines-P1/blob/master/writeup_template.md
4. Canny function: https://blog.csdn.net/sunny2038/article/details/9202641
5. 影片下載 https://oliver88.com/%E7%B6%B2%E8%B7%AF%E8%B3%87%E6%BA%90/youtube-video-download/
6. 原影片網址: https://www.youtube.com/watch?v=kS9Cm4vXxho