車道偏離及安全距離偵測

組員：

S1154007 賴宥瑋

S1154008 林芷瑩

S1154043 陳睿彰

一、設計理念

隨著車輛數量的持續增加，交通事故也越來越頻繁，其中一項法律條文「應注意而未注意」在實務上常令人感到模糊，缺乏明確的標準。為了提升行車安全，我們計劃設計一套精確的車距檢測系統。該系統將能有效辨識不同類型的車輛，如摩托車與小汽車，並準確估算它們與自身車輛之間的距離。當系統偵測到潛在的安全風險時，會立即發出警示，幫助駕駛者在危險發生前迅速反應。我們希望透過這項技術，有效縮短反應時間或預留足夠的安全距離，從而降低事故發生的可能性，為駕駛者提供更高的行車安全保障。

二、技術選擇與設計概述

本系統採用 YOLOv11 模型進行物體偵測，該模型能在實時環境中快速、準確地識別多種類型的物體。為專注於目標車輛，我們對 YOLOv11 模型進行微調，使其僅檢測摩托車和小汽車，過濾掉其他不相關物體。距離估算則基於攝影測量公式，結合車輛實際寬度、相機焦距與物體邊界框尺寸進行計算，提供精確的實時距離估算結果。

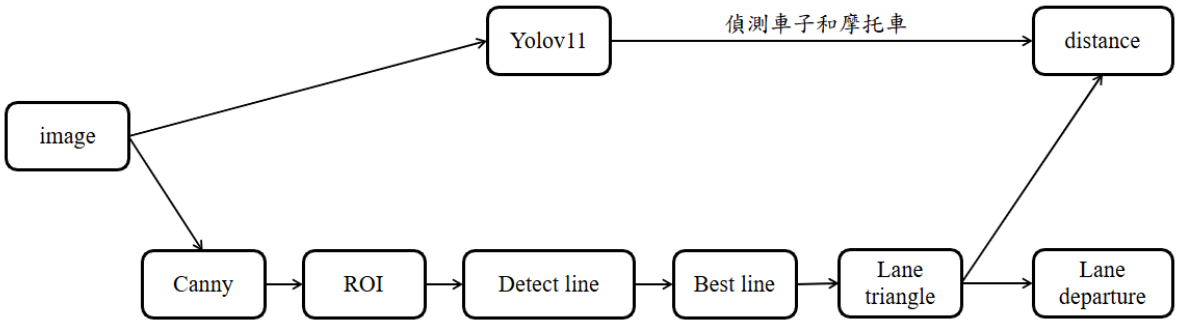
- 設計原則

1. 準確性：確保摩托車與小汽車的檢測準確性，防止錯誤分類或漏檢。
2. 即時性：系統應具備高速處理能力，以滿足高速公路監控需求，因此選用輕量化的 YOLOv11 模型並進行優化。
3. 擴展性：考慮未來可能新增其他車輛類型的檢測需求，系統在設計時注重可擴展性，便於後續升級與擴展。

- 模組設計與實現

1. 物體檢測模組：利用 YOLOv11 模型檢測圖像中的摩托車與小汽車的邊界框，輸出車輛類型及其位置資訊。
2. 距離估算模組：根據物體邊界框尺寸、預設焦距值與車輛實際寬度，使用距離估算公式計算車輛與相機之間的距離。
3. 結果輸出模組：實時顯示車輛類型與距離資訊，並為後續的車速計算或交通監測系統提供數據支持。

- 流程圖



三、設計細節

- 道路偏移

1. Canny：將輸入圖像進行灰階轉換、模糊處理，再使用 Canny 邊緣檢測方法提取圖像中的邊緣。

```
def canny_edge_detection(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)   # 轉為灰階
    blur = cv2.GaussianBlur(gray, (5, 5), 0)         # 高斯模糊降噪
    edges = cv2.Canny(blur, 50, 100)                 # Canny 邊緣檢測
    return edges
```

2. ROI (region of interest)：提取圖像中的特定區域，只針對感興趣的地方偵測。通過定義一個梯形的 ROI，過濾掉無關部分，提高後續圖像處理的準確性和效率。

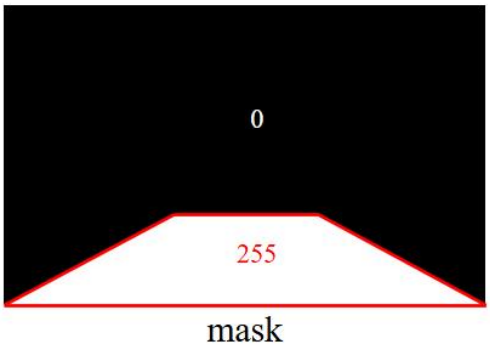
ROI區域

(1) polygon 定義了一個多邊形區域，用於描述 ROI 的頂點。

(2) mask 是與原圖像大小相同的黑色圖像，所有像素值初始化為 0 (黑色)。

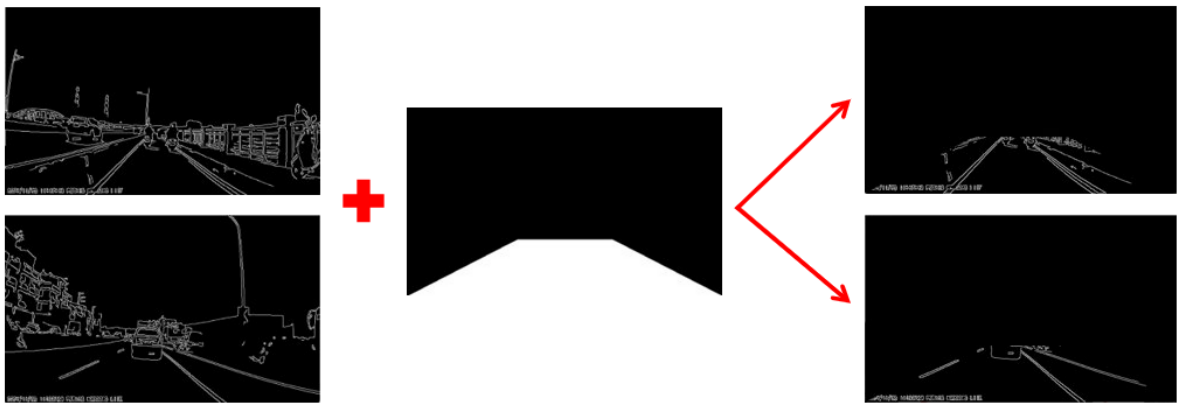
(3) 使用 cv2.fillPoly 在 mask 上繪製多邊形，並將該區域填充為白色 (255)。

```python
# 定義多邊形mask的四個頂點
polygon = np.array([[
    (int(width * 0.65), int(height * 0.7)),  # 右上角
    (int(width * 0.35), int(height * 0.7)),  # 左上角
    (int(width * 0.0), height),              # 左下角
    (width, height)                          # 右下角
]], dtype=np.int32)

# 建立空白mask
mask = np.zeros_like(image)

# 在mask上填充梯形區域為白色
cv2.fillPoly(mask, polygon, 255)
```



(4) 使用 cv2.bitwise_and 將原圖與 mask 合併，只有 mask 為白色的區域會保留，其他區域變為黑色。

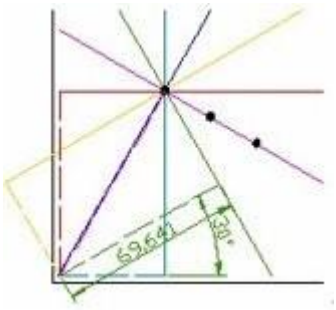
```python
masked_image = cv2.bitwise_and(image, mask)
```



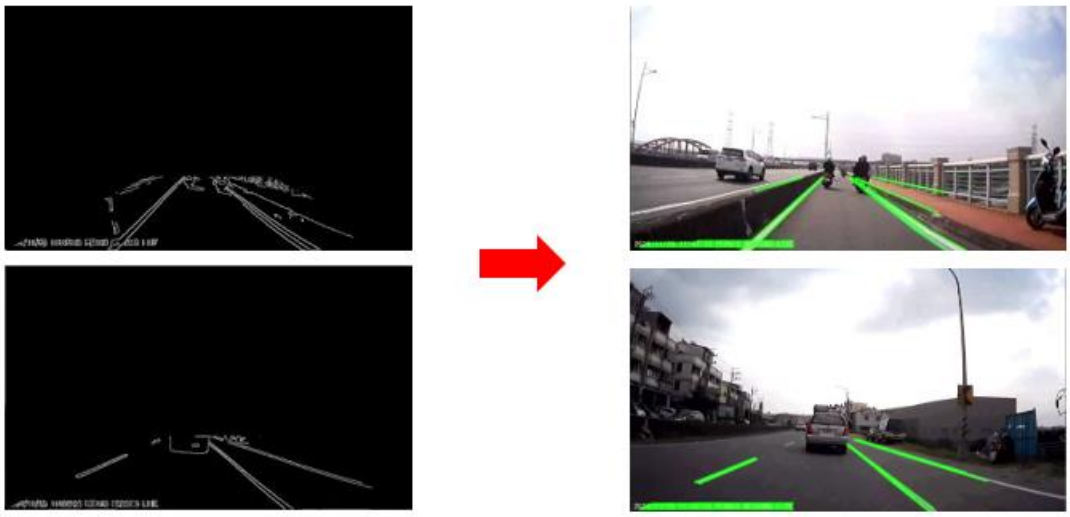
3. detect line：使用霍夫線變換檢測直線段。

lines = **cv2.HoughLinesP**(image, rho, theta, threshold, minLineLength, maxLineGap)

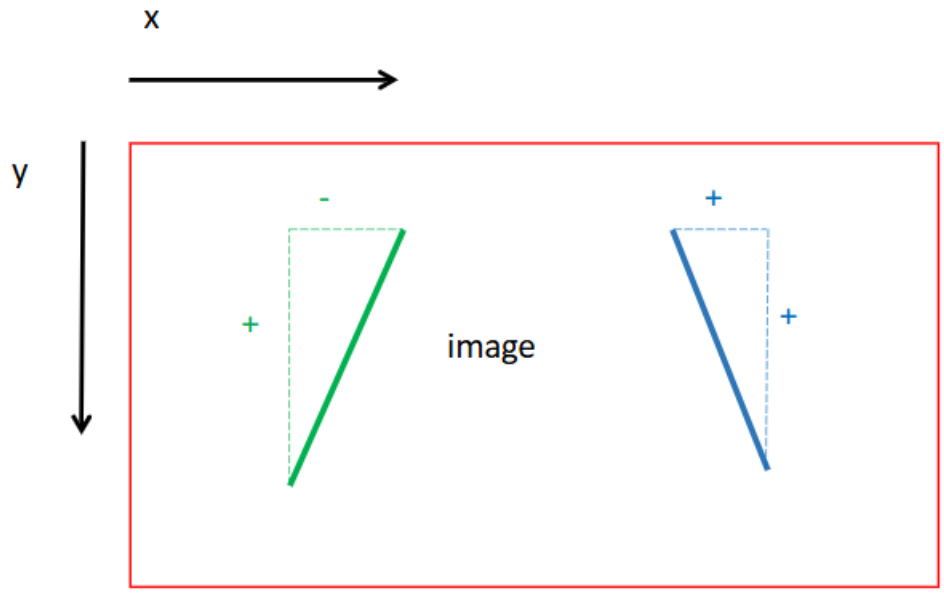
(1) image：必須是 binary image，canny 後的 。
(2) rho: 線段以像素為單位的距離精度 。
(3) theta：線段以弧度為單位的角度精度。
(4) threshod: 累加平面的閾值參數最小要超過多少 。
(5) minLineLength： 線段以像素為單位最小要超過多少。
(6) maxLineGap： 同一方向上兩條線要不要連成一條。



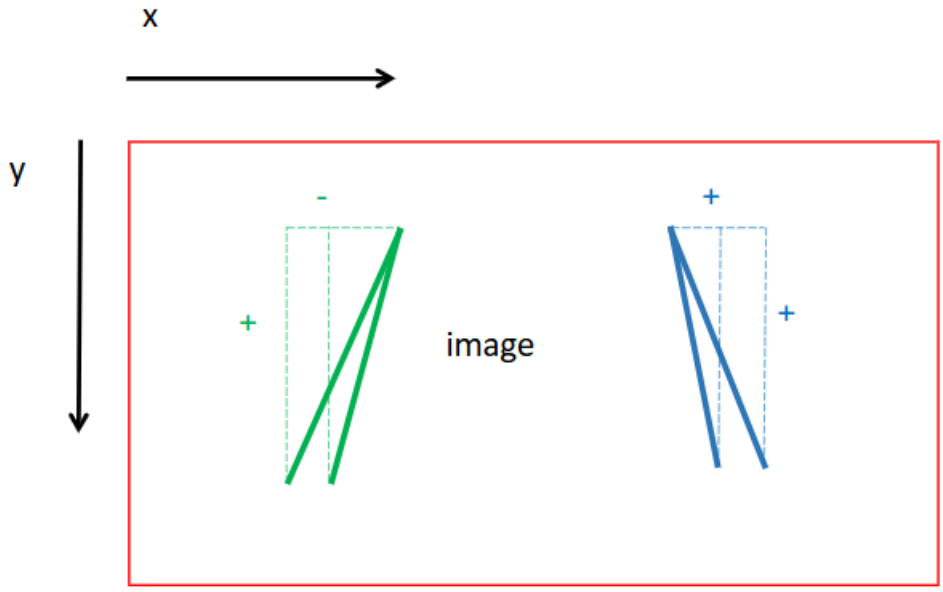
```
lines = cv2.HoughLinesP(cropped_edges, rho=1, theta=np.pi / 180, threshold=100,
                        minLineLength=100, maxLineGap=50)
```



4. best line：根據檢測到的直線擬合左車道線和右車道線，選擇斜率最大的直線。
   (1) $m > 0$: 右車道 ，$m < 0$: 左車道。



(2) 將所有左右車道候選人存起來，取絕對值最大的當作 best Line。



(3) 遍歷直線並計算斜率與截距。

- 提取直線端點座標：x1、y1、x2、y2。
- 避免垂直線：檢查 x2 - x1 是否為零，以免斜率計算時發生除以零錯誤。
- 過濾平緩線條：若斜率的絕對值小於 0.3，視為幾乎水平的線條，不予考慮。
- 篩選左車道線：斜率為負，且線段的 x 座標位於圖像左側（x2 <= 0.51 * width）。
- 篩選右車道線：斜率為正，且線段的 x 座標位於圖像右側（x2 >= 0.49 * width）。

```
for line in lines:
    x1, y1, x2, y2 = line[0]
    # 檢查是否有垂直的直線（防止除以零）
    if x2 - x1 != 0:
        slope = (y2 - y1) / (x2 - x1)  # 計算斜率
        intercept = y1 - slope * x1
        if abs(slope) > 0.3:  # 過濾掉幾乎水平的線條
            if slope < 0 and x2 <= 0.51 * width:   # 左車道線（斜率為負）
                left_lines.append((slope, intercept))
            elif slope > 0 and x2 >= 0.49 * width:  # 右車道線（斜率為正）
                right_lines.append((slope, intercept))
```

(4) 在存取的所有車道線中，選擇絕對值最大也就是斜率最大的車道線。

```
# 選擇斜率最大的右車道線
if right_lines:
    max_slope_line = right_lines[0]
    max_slope_value = abs(max_slope_line[0])
    for line in right_lines[1:]:
        slope = abs(line[0])
        if slope > max_slope_value:
            max_slope_line = line
            max_slope_value = slope
    right_lines = [max_slope_line]
```

```
# 選擇斜率最大的左車道線
if left_lines:
    max_slope_line = left_lines[0]
    max_slope_value = abs(max_slope_line[0])
    for line in left_lines[1:]:
        slope = abs(line[0])
        if slope > max_slope_value:
            max_slope_line = line
            max_slope_value = slope
    left_lines = [max_slope_line]
```

5. Lane triangle：

(1) 計算車道線的交點（通常位於圖像的上部）。

(2) 計算車道線在圖像底部的 x 座標。

(3) 定義一個三角形，包含左車道底部點、右車道底部點和車道線交點，用於描述車輛行駛的車道區域。

左車道線方程：y = left_slope·x + left_intercept
右車道線方程：y = right_slope·x + right_intercept

交點計算:
left_slope·x + left_intercept = right_slope·x + right_intercept
x ·(left_slope−right_slope) = right_intercept − left_intercept

$$x = \frac{\text{right\_intercept} - \text{left\_intercept}}{\text{left\_slope} - \text{right\_slope}}$$

y = left_slope·x + left_intercept

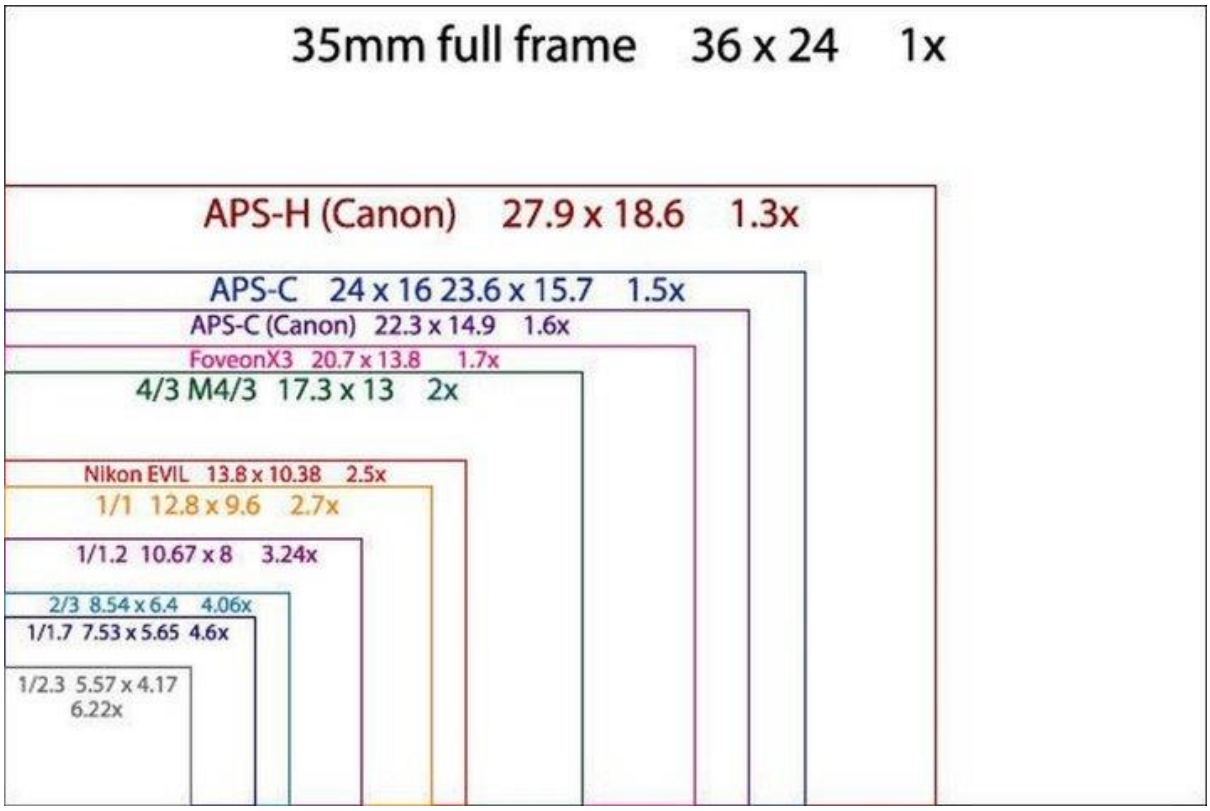
```
left_slope, left_intercept = left_lines[0]
right_slope, right_intercept = right_lines[0]

# 計算交點
x_inter = int((right_intercept - left_intercept) / (left_slope - right_slope))
y_inter = int(left_slope * x_inter + left_intercept)

# 計算左右線的底部的 x 座標
x_left_bottom  = int((y_bottom - left_intercept) / left_slope)

x_right_bottom = int((y_bottom - right_intercept) / right_slope)

# 定義三角形的三個頂點
triangle_points = np.array([
    [x_left_bottom, y_bottom],    # 左底部
    [x_right_bottom, y_bottom],   # 右底部
    [x_inter, y_inter]            # 左右線的交點
], dtype=np.int32)
```

6. Lane departure：以圖像底部中心作為參考點，計算其到車道線的水平距離，判斷是否超出設定的容許範圍。如果車輛接近或碰觸車道線，提醒車輛可能偏離車道。

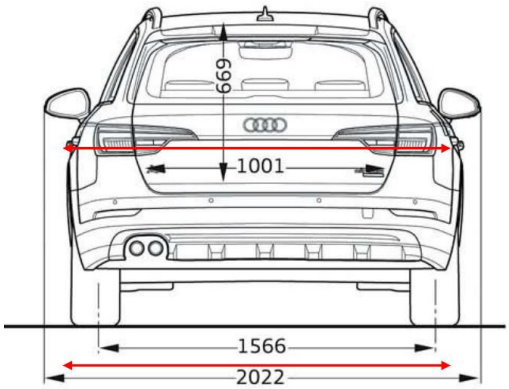
```python
# 定義底部中間的參考點
center_point = (int(width * 0.5), height - 10)

direction = "Lane Normal"  # 預設為正常狀態

# 設定容許誤差 (視為碰到車道線)
epsilon = 10

# 檢查是否碰到左車道線
if left_lines:
    slope, intercept = left_lines[0]
    x_at_line = (center_point[1] - intercept) / slope
    distance = abs(center_point[0] - x_at_line)
    if distance <= epsilon:
        direction = "Lane Departure : Left"

# 檢查是否碰到右車道線
if right_lines:
    slope, intercept = right_lines[0]
    x_at_line = (center_point[1] - intercept) / slope
    distance = abs(center_point[0] - x_at_line)
    if distance <= epsilon:
        direction = "Lane Departure : Right"
```

- 距離偵測
  1. 像素轉換：焦距（像素） =(焦距（毫米） / 感測器尺寸（毫米）)× 影像解析度（像素）。

     等效焦距 ：由於是行車紀錄器，所以在焦距失準之前，只需採用能在 5~30 米內準確度極高的數值便可，另外可通過調節數據使其應用在其他設備上。



2. 車輛寬度： 汽車對應 1.8(m)，機車對應 0.7(m) 。



3. 使用 YOLOv11 模型：使用 YOLOv11 模型來檢測車輛 ，僅處理汽車（類別 2）和機車（類別 3）。因為只處理兩個類別的物體 ，所以在過濾檢測結果，檢測邊界框部分確認物體類別後，將篩選範圍限制在汽車（類別 2）和機車（類別 3）。

```
# 專注於機車和汽車的檢測類別
vehicle_classes = [2, 3]  # YOLOv11中，2是車(car)，3是機車(motorbike)
```

```
# 只處理汽車和機車類別
if cls in vehicle_classes and is_bbox_in_lane(xmin, ymin, xmax, ymax, lane_polygon):
    if cls == 2:  # 汽車
        real_vehicle_width = 1.8  #  汽車的平均寬度約 1.8 米
    elif cls == 3:  # 機車
        real_vehicle_width = 0.7  # 假設機車的平均寬度約0.7 米
```

4. 距離估算：使用了一個典型的攝影測量公式來計算距離，這個公式基於車輛在畫面中的邊界框寬度（bbox width）以及相機的焦距（focal_length）和車輛實際寬度（real_vehicle_width）。

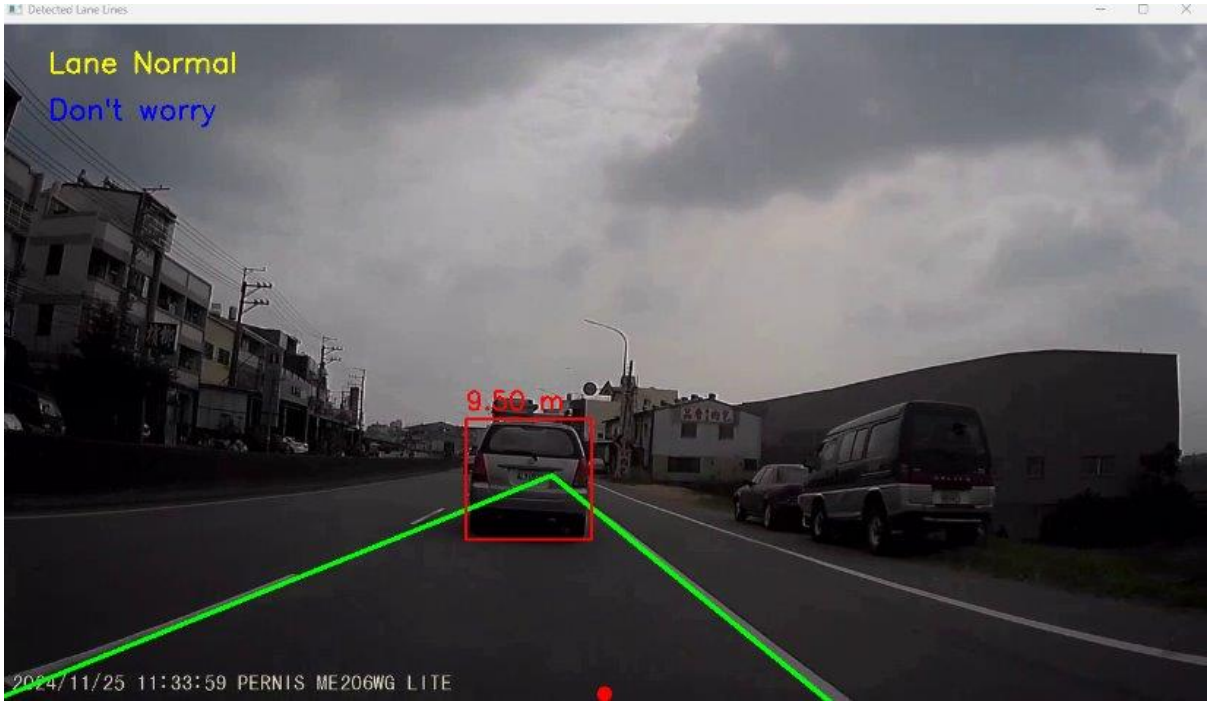
當車輛在影像中顯得越大（即 bbox_width_pixels 越大），計算出的距離會越小，表明車輛離攝像頭較近 ；當車輛在影像中顯得越小（即 bbox_width_pixels 越小），距離會越大，表示車輛距離攝像頭較遠。

```
def estimate_distance(bbox_width_pixels, focal_length, real_vehicle_width):
    return (focal_length * real_vehicle_width) / bbox_width_pixels
```

5. 安全顯示：當車輛在車道區域內且距離小於 20 米時，會在車輛周圍繪製紅色框，並顯示距離；如果車輛距離小於 4 米，會顯示“Notice”的提示，提醒駕駛員注意前方車輛過近 。

```
# 當車輛距離小於20米時顯示紅色框並顯示距離
if distance < 20:
    cv2.rectangle(frame, (int(xmin), int(ymin)), (int(xmax), int(ymax)), (0, 0, 255), 2)
    cv2.putText(frame, f"{distance:.2f} m", (int(xmin), int(ymin) - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)
else:
    cv2.rectangle(frame, (int(xmin), int(ymin)), (int(xmax), int(ymax)), (255, 0, 0), 2)

# 如果距離小於4米顯示Notice，並且避免重複顯示
if distance < 4 and not notice_shown:
    cv2.putText(frame, "Notice", (50, 100), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)
    notice_shown = True  # 顯示後將標誌設置為True
```

6. 比較實際距離：用白線以及間隔粗估距離。如下圖，白線為 4 公尺，白線間隔 6 公尺，則實際偵測到的距離約為 9.50 公尺。



四、困難與挑戰

在偵測車道線的時候，ROI 和 houghlinesp 的參數很難定義。

ROI 是經過很多次調整後，才找到最適合的圖形去過濾。而 houghlinesp 儘管在 ROI 過濾後也有可能偵測到雜訊，比如人的褲子、摩托車的邊緣等都有可能是直線的候選或者是成為車道線。而我們的做法是改變 houghlinesp 裡的參數，就可以解決了。

五、結論

本系統以提高行車安全為核心，結合 YOLOv11 進行高效的車輛偵測與距離估算，並應用圖像處理技術準確辨識車道線和車道偏移狀態。 未來，此系統具備擴展性，可應用於更廣泛的車輛類型與場景，為駕駛者提供更智能的安全輔助功能，有效降低交通事故風險。