**EG114728 HD in AI and Robotics**

**MBS4544 Robot Sensing and Vision**

**Assignment 4 (20%)**

**Requirement:**

* **Individual submission**
* **Submit a pdf or word file with all codes, screen shots and texts**
* **Plagiarism will result in ZERO mark**
* **Deadline: 18 Dec 2023**

**Question 1 (8%)**

Briefly explain the theory and arithmetic behind the Hough Line Transform and state two applications that use Hough Transform.

The Hough Line Transform is a technique used in computer vision and image processing to detect lines in an image. It was introduced by Paul Hough in 1962. The basic idea is to represent lines in the image space as points in a parameter space (Hough space) and then identify lines by finding peaks in the accumulator space.

The standard Hough Line Transform is applied to detect straight lines. For each edge pixel in the image, it votes in the Hough space for possible lines that could pass through it. The parameters of the lines are typically represented as slope-intercept pairs (m, b) or polar coordinates (r, θ). The accumulator space is a 2D array where each cell represents a possible line, and the cell value corresponds to the number of votes or points that support that line.

The steps of the Hough Line Transform can be summarized as follows:

* Edge detection: Detect edges in the image using techniques like the Canny edge detector.
* Hough space representation: For each edge point, convert its coordinates into the parameter space (Hough space).
* Accumulation: Accumulate votes in the Hough space for possible lines.
* Thresholding: Set a threshold in the accumulator space to identify lines with a sufficient number of votes.
* Line extraction: Extract the lines corresponding to the peaks in the accumulator space.

The Hough Line Transform has many applications in image analysis, computer vision, and digital image processing. Some examples are: Medical Image Analysis, Lane detection in autonomous driving

**Question 2 (12%)**

In the lecture, you learn how to detect the left and right lines on the road from the static image given (***lane\_detect.jpg***). Below results shown the steps of how to merge the polylines onto the original image.

A blue line on a black background

Description automatically generated with low confidence A picture containing shape

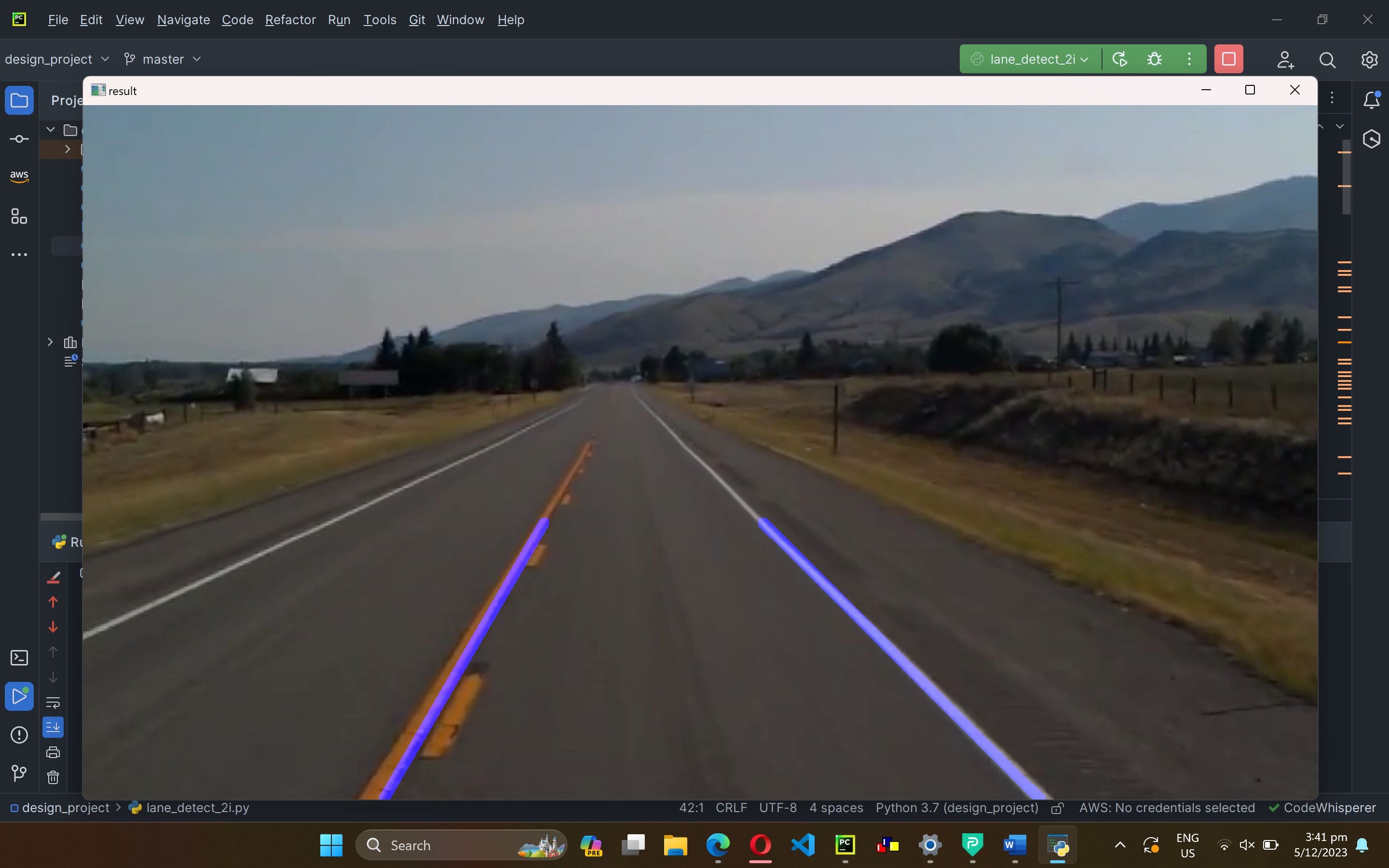
Description automatically generated

**A picture containing text, way, road, scene

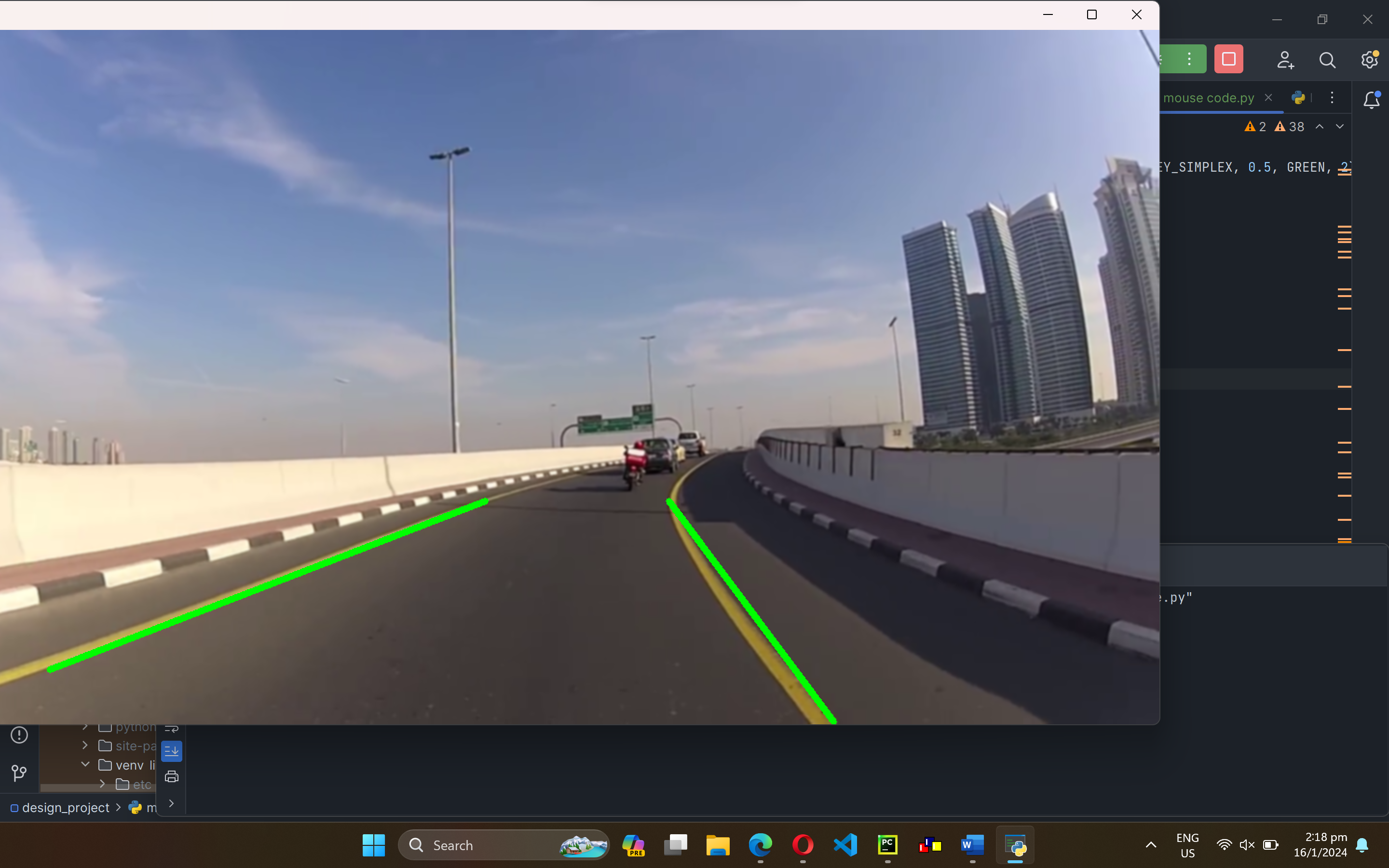
Description automatically generated**

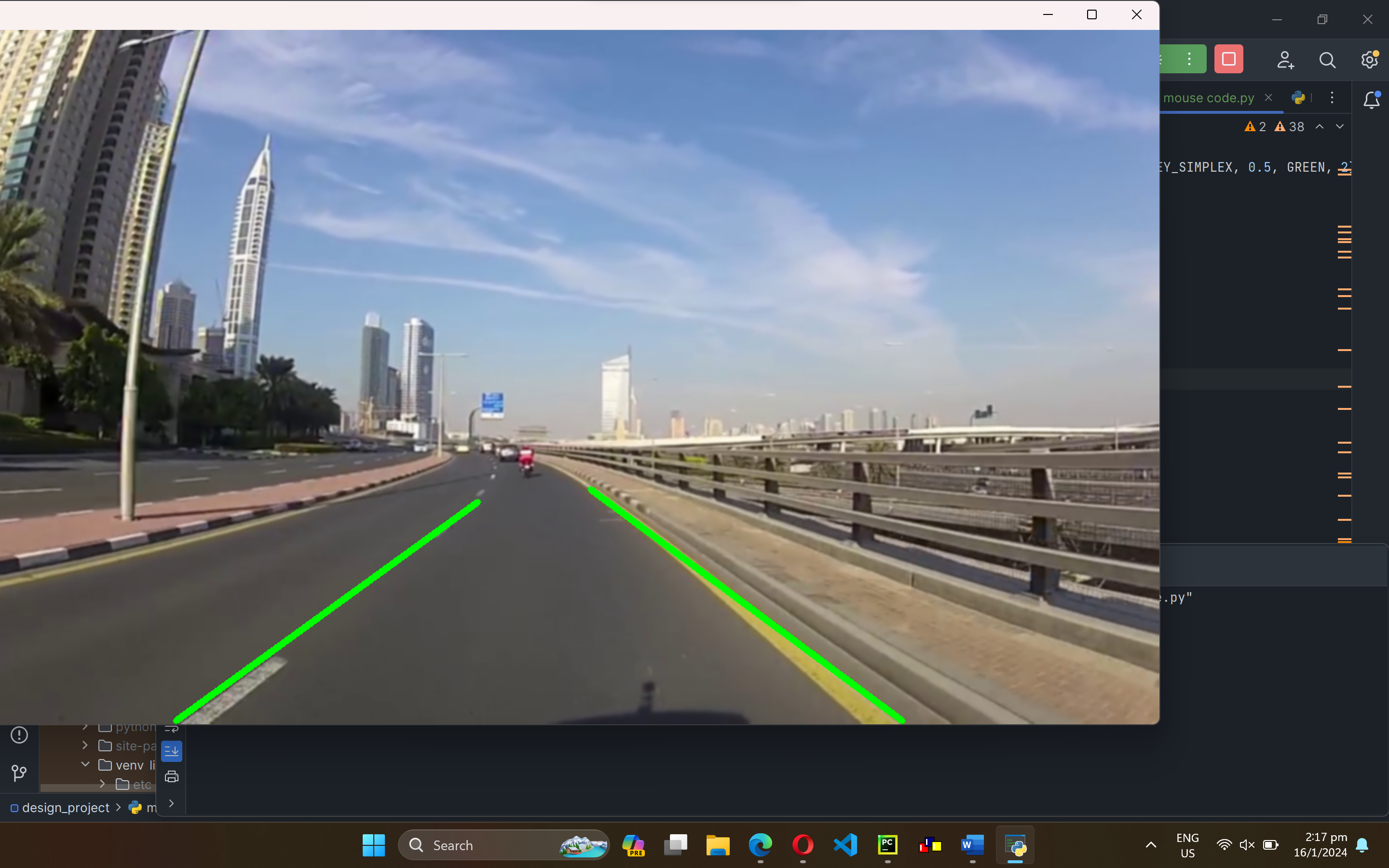
**Work to do:**

1. With the code that you have successfully tested on the static image above, try to apply on a video clip (***test2.mp4***) which is taken from the same site above.
   1. Submit the code
2. import cv2  
   import numpy as np  
     
   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])  
     
   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)  
    slope = parameters[0]  
    intercept = parameters[1]  
    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)  
    left\_line = make\_coordinates(image, left\_fit\_average)  
    right\_line = make\_coordinates(image, right\_fit\_average)  
    return np.array([left\_line, right\_line])  
     
   def canny(image):  
    gray = cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY)  
    blur = cv2.GaussianBlur(gray,(5, 5), 0)  
    canny = cv2.Canny(blur, 50, 150)  
    return canny  
     
   def display\_lines(image, lines):  
    line\_image = np.zeros\_like(image)  
    if lines is not None:  
    for x1, y1, x2, y2 in lines:  
    cv2.line(line\_image, (x1, y1), (x2, y2), (255, 0, 0), 10)  
    return line\_image  
     
   def region\_of\_interest(image):  
    height = image.shape[0]  
    polygons = np.array([  
    [(200, height), (1100, height), (550, 250)]  
    ])  
    mask = np.zeros\_like(image)  
    cv2.fillPoly(mask, polygons, 255)  
    masked\_image = cv2.bitwise\_and(image, mask)  
    return masked\_image  
     
   #image = cv2.imread('lane.jpg')  
   #lane\_image = np.copy(image)  
     
   #canny\_image = canny(lane\_image)  
   #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(lane\_image, lines)  
   #line\_image = display\_lines(lane\_image, averaged\_lines)  
     
   #combo\_image = cv2.addWeighted(lane\_image, 0.8, line\_image, 1, 1)  
     
   #cv2.imshow('result',combo\_image)  
   #cv2.waitKey(0)  
     
   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\_image = cv2.addWeighted(frame, 0.8, line\_image, 1, 1)  
    cv2.imshow('result',combo\_image)  
    if cv2.waitKey(1) == ord('q'):  
    break  
   cap.release()  
   cv2.destroyAllWindows()
   1. Submit a screen capture shot



1. Use the video clip given from last session (***road\_car\_view.mp4***), tweak the parameters and modify the code such that the correct lane will be detected which is similar to below pics.
   1. Submit the code
2. import cv2  
   import numpy as np  
   import math  
     
   video = cv2.VideoCapture("road\_car\_view.mp4")  
   GREEN = (0, 255, 0)  
   THICKNESS = 5  
   coordinates = []  
     
   class HoughBundler:  
    def get\_orientation(self, line):  
    orientation = math.atan2(abs((line[0] - line[2])), abs((line[1] - line[3])))  
    return math.degrees(orientation)  
     
    def checker(self, line\_new, groups, min\_distance\_to\_merge, min\_angle\_to\_merge):  
    for group in groups:  
    for line\_old in group:  
    if self.get\_distance(line\_old, line\_new) < min\_distance\_to\_merge:  
    orientation\_new = self.get\_orientation(line\_new)  
    orientation\_old = self.get\_orientation(line\_old)  
    if abs(orientation\_new - orientation\_old) < min\_angle\_to\_merge:  
    group.append(line\_new)  
    return False  
    return True  
     
    def DistancePointLine(self, point, line):  
    px, py = point  
    x1, y1, x2, y2 = line  
     
    def lineMagnitude(x1, y1, x2, y2):  
    lineMagnitude = math.sqrt(math.pow((x2 - x1), 2) + math.pow((y2 - y1), 2))  
    return lineMagnitude  
     
    LineMag = lineMagnitude(x1, y1, x2, y2)  
    if LineMag < 0.00000001:  
    DistancePointLine = 9999  
    return DistancePointLine  
     
    u1 = (((px - x1) \* (x2 - x1)) + ((py - y1) \* (y2 - y1)))  
    u = u1 / (LineMag \* LineMag)  
     
    if (u < 0.00001) or (u > 1):  
    ix = lineMagnitude(px, py, x1, y1)  
    iy = lineMagnitude(px, py, x2, y2)  
    if ix > iy:  
    DistancePointLine = iy  
    else:  
    DistancePointLine = ix  
    else:  
    ix = x1 + u \* (x2 - x1)  
    iy = y1 + u \* (y2 - y1)  
    DistancePointLine = lineMagnitude(px, py, ix, iy)  
     
    return DistancePointLine  
     
    def get\_distance(self, a\_line, b\_line):  
    dist1 = self.DistancePointLine(a\_line[:2], b\_line)  
    dist2 = self.DistancePointLine(a\_line[2:], b\_line)  
    dist3 = self.DistancePointLine(b\_line[:2], a\_line)  
    dist4 = self.DistancePointLine(b\_line[2:], a\_line)  
     
    return min(dist1, dist2, dist3, dist4)  
     
    def merge\_lines\_pipeline\_2(self, lines):  
    groups = []  
    min\_distance\_to\_merge = 100  
    min\_angle\_to\_merge = 100  
    groups.append([lines[0]])  
    for line\_new in lines[1:]:  
    if self.checker(line\_new, groups, min\_distance\_to\_merge, min\_angle\_to\_merge):  
    groups.append([line\_new])  
     
    return groups  
     
    def merge\_lines\_segments1(self, lines):  
    orientation = self.get\_orientation(lines[0])  
     
    if len(lines) == 1:  
    return [lines[0][:2], lines[0][2:]]  
     
    points = []  
    for line in lines:  
    points.append(line[:2])  
    points.append(line[2:])  
    if 0 < orientation < 0:  
    points = sorted(points, key=lambda point: point[1])  
    else:  
    points = sorted(points, key=lambda point: point[0])  
     
    return [points[0], points[-1]]  
     
    def process\_lines(self, lines, img):  
    lines\_x = []  
    lines\_y = []  
    for line\_i in [l[0] for l in lines]:  
    orientation = self.get\_orientation(line\_i)  
    if 0 < orientation < 0:  
    lines\_y.append(line\_i)  
    else:  
    lines\_x.append(line\_i)  
     
    lines\_y = sorted(lines\_y, key=lambda line: line[1])  
    lines\_x = sorted(lines\_x, key=lambda line: line[0])  
    merged\_lines\_all = []  
     
    for i in [lines\_x, lines\_y]:  
    if len(i) > 0:  
    groups = self.merge\_lines\_pipeline\_2(i)  
    merged\_lines = []  
    for group in groups:  
    merged\_lines.append(self.merge\_lines\_segments1(group))  
    merged\_lines\_all.extend(merged\_lines)  
     
    return merged\_lines\_all  
     
   def region\_of\_interest(image):  
    height = image.shape[0]  
    width = image.shape[1]  
    polygons = np.array([  
    [(0, height), (width, height), (700, 450), (500, 500)]  
    ])  
    mask = np.zeros\_like(image)  
    cv2.fillPoly(mask, polygons, 255)  
    masked\_image = cv2.bitwise\_and(image, mask)  
    return masked\_image  
     
     
   def on\_mouse\_click(event, x, y, flags, param):  
    if event == cv2.EVENT\_LBUTTONDOWN:  
    coordinates.append((x, y))  
    print(f"Clicked at (x, y): ({x}, {y})")  
    cv2.putText(frame, f"Clicked at (x, y): ({x}, {y})", (x, y), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, GREEN, 2)  
     
   cv2.namedWindow("frame")  
   cv2.setMouseCallback("frame", on\_mouse\_click)  
     
   while video.isOpened():  
    ret, orig\_frame = video.read()  
    if not ret:  
    video = cv2.VideoCapture("road\_car\_view.mp4")  
    continue  
    frame = cv2.GaussianBlur(orig\_frame, (5, 5), 0)  
     
    hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)  
    low\_yellow = np.array([18, 94, 105])  
    up\_yellow = np.array([48, 255, 255])  
    mask = cv2.inRange(hsv, low\_yellow, up\_yellow)  
    edges = cv2.Canny(mask, 75, 150)  
    cropped\_image = region\_of\_interest(edges)  
    lines = cv2.HoughLinesP(cropped\_image, 1, np.pi / 180, 50, maxLineGap=50)  
     
    if lines is not None:  
    bundler = HoughBundler()  
    merged\_lines = bundler.process\_lines(lines, edges)  
     
    if len(merged\_lines) == 1:  
    x1, y1 = map(int, merged\_lines[0][0])  
    x2, y2 = map(int, merged\_lines[0][1])  
    # d1, e1 = (618, 463)  
    # d2, e2 = (225, 677)  
     
    d1, e1 = (573, 489)  
    d2, e2 = (260, 716)  
    # (1271, 690)  
    # (886, 521)  
    if x1 > 500: # Green line on the left  
    cv2.line(frame, (d1, e1), (d2, e2), GREEN, THICKNESS)  
    else: # Green line on the right  
    d1, e1 = (804, 494)  
    d2, e2 = (1250, 688)  
    cv2.line(frame, (d1, e1), (d2, e2), GREEN, THICKNESS)  
     
    for line in merged\_lines:  
    x1, y1 = map(int, line[0])  
    x2, y2 = map(int, line[1])  
    cv2.line(frame, (x1, y1), (x2, y2), GREEN, THICKNESS)  
     
    #cv2.imshow("cropped\_image", cropped\_image)  
    cv2.imshow("frame", frame)  
     
     
    key = cv2.waitKey(1)  
    if key == 27:  
    break  
     
   video.release()  
   cv2.destroyAllWindows()
   1. Submit two screen capture shots (like those shown below)





**A highway with cars on it

Description automatically generated with low confidence**

****