Katherine Banis

CMPSC 497

Lab #8: Lane Detection with OpenCV and Python

10/22/2024

Objective: Using real world images of roads and streets, create a program using Python and OpenCV to display each edge line with a red line and print out eh angle of the line corresponding to the edge of the road.

Materials: Python, OpenCV, phone camera

Pseudocode:

1. canny: Function for canny edge detection
   1. Input: image
   2. Convert image to grayscale
   3. Apply gaussian blue to reduce noise
   4. Perform canny edge detection to find edges
   5. Return edge-detected image
2. display\_lines: Function to display lines and calculate angles
   1. Input: image and detected lines
   2. Create an empty image with same size as input images
   3. Initialize lists to store parameters (slope and intercept) for left and right lanes
   4. If any lines are detected:
      1. For each line:
         1. Extract coordinates of line
         2. Calculate slope and intercept of line
         3. If slope is negative: classify line as left lane
         4. If slope is positive: classify line as right lane
   5. coordinates: Helper function to calculate coordinates of line using slope and intercept
   6. print\_angle: Helper function to calculate and print the angle of the lane line using slope.
   7. If left lane is detected:
      1. Calculate average slope and intercept for left lane
      2. Calculate coordinates for left lane line
      3. Draw left lane line on image
      4. Print angle of left lane line
   8. If right lane is detected:
      1. Calculate average slope and intercept for right lane
      2. Calculate coordinates for right lane line
      3. Draw right lane on the image
      4. Print angle of right lane line
   9. Return image with both lane lines drawn
3. region\_of\_interest: Function for region of interest (ROI) mask
   1. Input: image
   2. Get the height and width of an image
   3. Define a polygon covering the area of interest using the width and height
   4. Create a mask that is black everywhere except inside ROI
   5. Apply mask to image to retain only region inside ROI
   6. Return masked image
4. process\_image: Function to process the image
   1. Input: path to image
   2. Load image from file
   3. If image fails to load: print error message and exit program
   4. Create a copy of image for lane detection
   5. Apply edge detection using canny function
   6. Apply ROI mask using region\_of\_interest function
   7. Detect lines in masked image using hough line transform
   8. Draw detected lane lines on image using display\_lines function
   9. Combine original image and the image with lane lines
   10. Display final image with lane lines and angles
   11. Close display window
5. Call process\_image function with image path

Test Cases:

Test 1:

A road with trees and blue sky

Description automatically generated

Test 2:

A road with trees and power lines

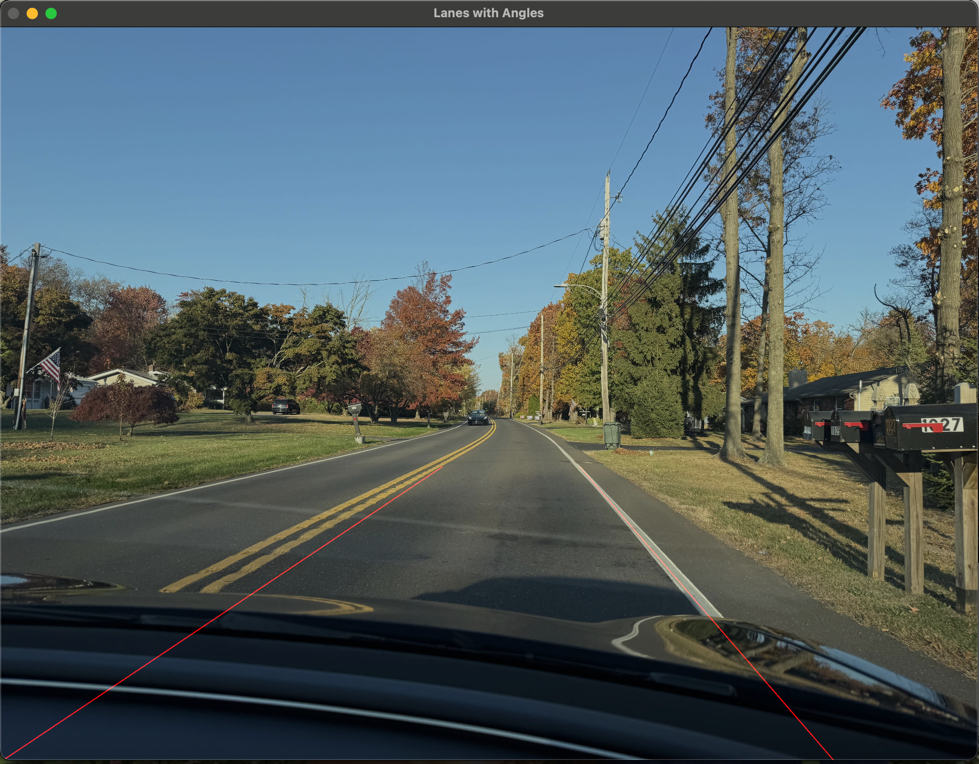
Description automatically generated

Results:

A road with trees in the background

Description automatically generated

Test 1 output: left lane line angle: -36.13 degrees.  
 right lane line angle: 35.16 degrees.



Test 2 output: left lane line angle: -33.76 degrees.  
 right lane line angle: 49.15 degrees.

Conclusion:

This was the most challenging homework for me so far. At first, I could only get the right edge to be detected. Then after making some modifications to the display\_lines function, it couldn’t detect any edges. I continuously went back to my original code to start over and work off of and eventually it was detecting everything like the trees. So, after lots of trial and error, I was finally successful but it doesn’t work with every image I tried. There is one image not featured in this report where I think the front of my car was too high up in the photo that it focused on that instead of the road. And it’s not perfect, the lines are very straight and don’t exactly follow the road so there is definitely room for improvement. But I learned a lot through this assignment.

Python Script:

import cv2 as cv  
import numpy as np  
  
  
# function to apply canny edge detection on image  
def canny(im):  
 # convert image to grayscale  
 gray = cv.cvtColor(im, cv.COLOR\_RGB2GRAY)  
 # apply gaussian blue to reduce noise and improve edge detection  
 blur = cv.GaussianBlur(gray, (5, 5), 0)  
 # perform canny edge detection  
 canny = cv.Canny(blur, 50, 150)  
 # return edge-detected image  
 return canny  
  
  
# function to draw detected lines on image  
def display\_lines(im, lines):  
 # create blank image with same dimensions as image  
 line\_im = np.zeros\_like(im)  
  
 # Lists to store left and right lane line parameters (slope, intercept)  
 left\_fit = []  
 right\_fit = []  
  
 # check if any lines are detected  
 if lines is not None:  
 # iterate over each line  
 for line in lines:  
 # each line is defined by two points (x1, y1) and (x2, y2)  
 x1, y1, x2, y2 = line.reshape(4)  
  
 # calculate slope and intercept of line  
 if x2 - x1 == 0: # avoid division by zero  
 continue  
 slope = (y2 - y1) / (x2 - x1)  
 intercept = y1 - slope \* x1  
  
 # separate left and right lanes based on slope  
 if slope < -0.1: # negative slope = left lane  
 left\_fit.append((slope, intercept))  
 elif slope > 0.1: # positive slope = right lane  
 right\_fit.append((slope, intercept))  
  
 # function to create coordinates from slope and intercept  
 def coordinates(slope\_intercept):  
 slope, intercept = slope\_intercept  
 y1 = im.shape[0] # y1 = height of image  
 y2 = int(y1 \* 0.6) # y2 = 60% up the image  
 if slope == 0: # Avoid division by zero  
 return None  
 # calculate x1 and x2: x = (y - intercept ) / slope  
 x1 = int((y1 - intercept) / slope)  
 x2 = int((y2 - intercept) / slope)  
 return np.array([x1, y1, x2, y2])  
  
 # function to calculate and print angle of lane line  
 def print\_angle(slope, label):  
 # calculate angle from slope  
 angle = np.degrees(np.arctan(slope))  
 print(f"{label} lane line angle: {angle:.2f} degrees.")  
  
 # calculate average slope and intercept for left lane  
 if left\_fit:  
 left\_avg = np.average(left\_fit, axis=0)  
 left\_line = coordinates(left\_avg)  
 # draw line on image  
 if left\_line is not None:  
 cv.line(line\_im, (left\_line[0], left\_line[1]), (left\_line[2], left\_line[3]), (0, 0, 255), 5)  
 # print angle of left lane  
 print\_angle(left\_avg[0], "left")  
  
 # calculate average slope and intercept for left lane  
 if right\_fit:  
 right\_avg = np.average(right\_fit, axis=0)  
 right\_line = coordinates(right\_avg)  
 # draw line on image  
 if right\_line is not None:  
 cv.line(line\_im, (right\_line[0], right\_line[1]), (right\_line[2], right\_line[3]), (0, 0, 255), 5)  
 # print angle of right lane  
 print\_angle(right\_avg[0], "right")  
  
 # return image with lane lines drawn on it  
 return line\_im  
  
  
# function to define and apply a region of interest (ROI) mask  
def region\_of\_interest(im):  
 # get height and width of image  
 height = im.shape[0]  
 width = im.shape[1]  
  
 # define polygon that will represent ROI  
 polygons = np.array([  
 [(int(width \* 0.05), height), # bottom left corner  
 (int(width \* 0.95), height), # bottom right corner  
 (int(width \* 0.5), int(height \* 0.55))] # top center point  
 ], np.int32) # ensure polygon coordinates are integers  
  
 # create a mask  
 mask = np.zeros\_like(im)  
 # fill mask with white inside polygon  
 cv.fillPoly(mask, polygons, 255)  
 # apply mask to image using bitwise AND operation  
 masked\_im = cv.bitwise\_and(im, mask)  
  
 # return masked image  
 return masked\_im  
  
  
# function to process an image  
def process\_image(image\_path):  
 # Load the image of the road  
 im = cv.imread(image\_path)  
 if im is None:  
 print(f"Error: Unable to open image file '{image\_path}'.")  
 return  
  
 lane\_im = np.copy(im)  
  
 # Apply edge detection and region masking  
 canny\_im = canny(lane\_im)  
 cropped\_im = region\_of\_interest(canny\_im)  
  
 # Detect lines using Hough Transform  
 lines = cv.HoughLinesP(cropped\_im, 2, np.pi / 180, 100, np.array([]), minLineLength=30, maxLineGap=150)  
  
 # Display lines and angles  
 line\_im = display\_lines(lane\_im, lines)  
  
 # Combine original image with line image  
 combine\_im = cv.addWeighted(lane\_im, 0.8, line\_im, 1, 1)  
  
 # Show result  
 cv.imshow("Lanes with Angles", combine\_im)  
 cv.waitKey(0)  
 cv.destroyAllWindows()  
  
  
# Test on the uploaded image  
process\_image('road2.jpg')