CS 3430: SciComp with Py
Assignment 8: Detecting Left and Right Lanes in Road Images

Vladimir Kulyukin
Department of Computer Science
Utah State University

1. Learning Objectives

1. Hough Transform
2. OpenCV
3. Line Detection
4. Programming RPi with Python
5. Generators

2. Introduction

In this assignment, we will explore how Hough Transform can be used in lane detection, an important task in self-driving cars. **Figures 1** – **5** give several frames of Logan streets captured with the RPi in my Wrangler. In each frame, the blue lines mark detected left lanes and the red lines signify right lanes.

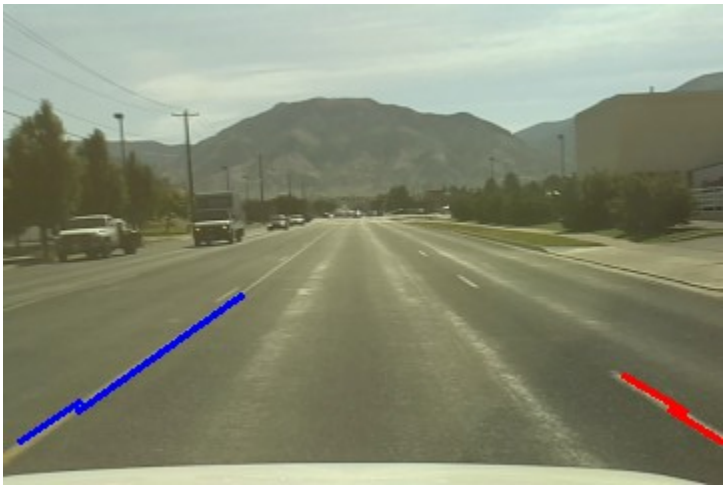

Figure 1. Left (blue) and right (red) lanes detected

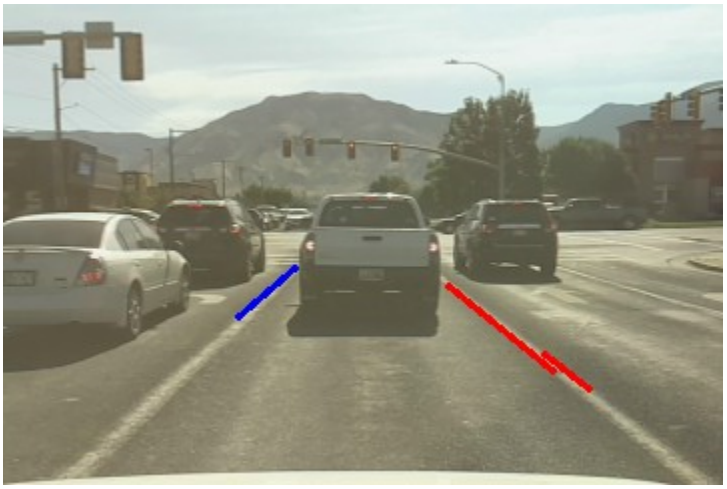

Figure 2. Both lanes are detected

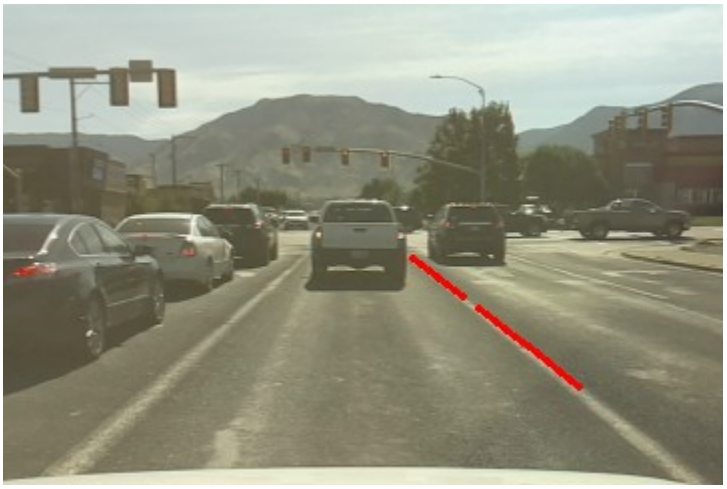

Figure 3. Only right lane detected

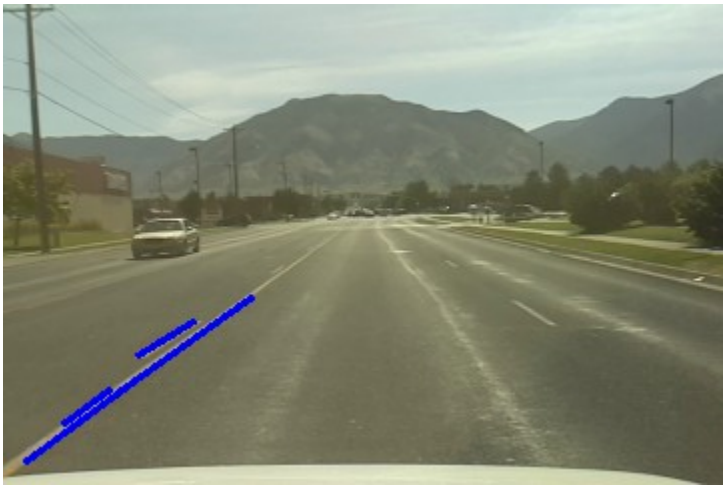

Figure 4. Only left lane detected

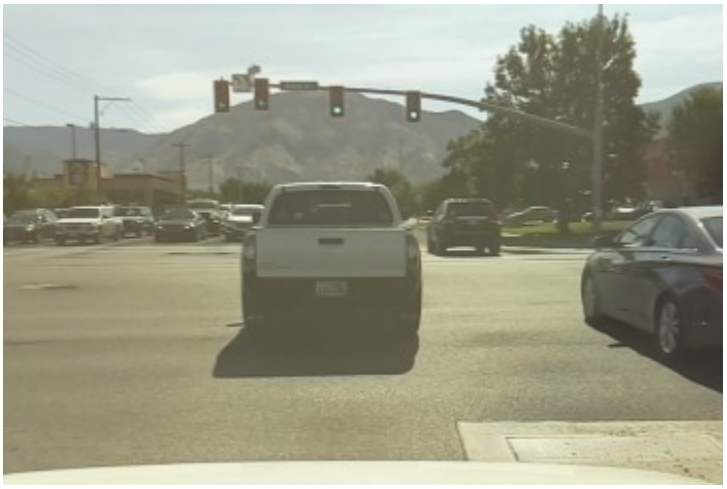

Figure 5. No lanes detected

Figures 1 and **2** illustrate cases when the CV system detects both lanes. In many captured frames, however, only one lane is detected, as shown in **Figures 4** and **5**. In some frames, no frames are detected, as in **Figure 5**. This case is a case of a true negative when the CV system should not detect any lanes, because none are present. False positives, i.e., frames when the CV system detects lanes when none are present, are much harder to deal with and cause more of a problem at turn time. The archive **road_images.zip** has 82 PNG images. You will write code to detect lanes in these images. The file **ht_detect_lanes.py** contains the stubs of all functions you will define in this assignment. And it is the only file you will need to submit through Canvas. You also have to code and run this assignment on your RPi.

1. Computing the Angle of a Line

Recall that in the lecture on Hough Transform, we discussed the method **cv2.HoughLinesP(image, rho_accuracy, theta_accuracy, num_votes, min_len, max_gap)** that returns a numpy array of lines detected in **image**. Each line is represented in as a list of two 2D points it passes through: **[x1, y1, x2, y2]**. The first two numbers are the coordinates of the first point and the second two numbers are the coordinates of the second point. Not every line detected in the image signifies the presence of a lane. Typically, it is only diagonal lines that signify lanes. Therefore, start this assignment by defining the function **line_deg_line(x1, y1, x2, y2)** that takes two 2D points and outputs the angle of the line in degrees. Radians would work just as well, but degrees are easier to deal with in debugging. A couple of examples of what this function outputs.

> line_deg_angle(1, 1, 5, 5)
45.0
> line_deg_angle(0, 5, 5, 0)
-45.0

The file **ht_detect_lanes.py** has the function **plot_ht_lanes_in_image** that takes a path to an image file and all the arguments to **cv2.HoughLinesP()** and draws all lines detected in the original image. It also displays the grayscale version of the images and all the edges detected in it, as shown in **Figure 6**.

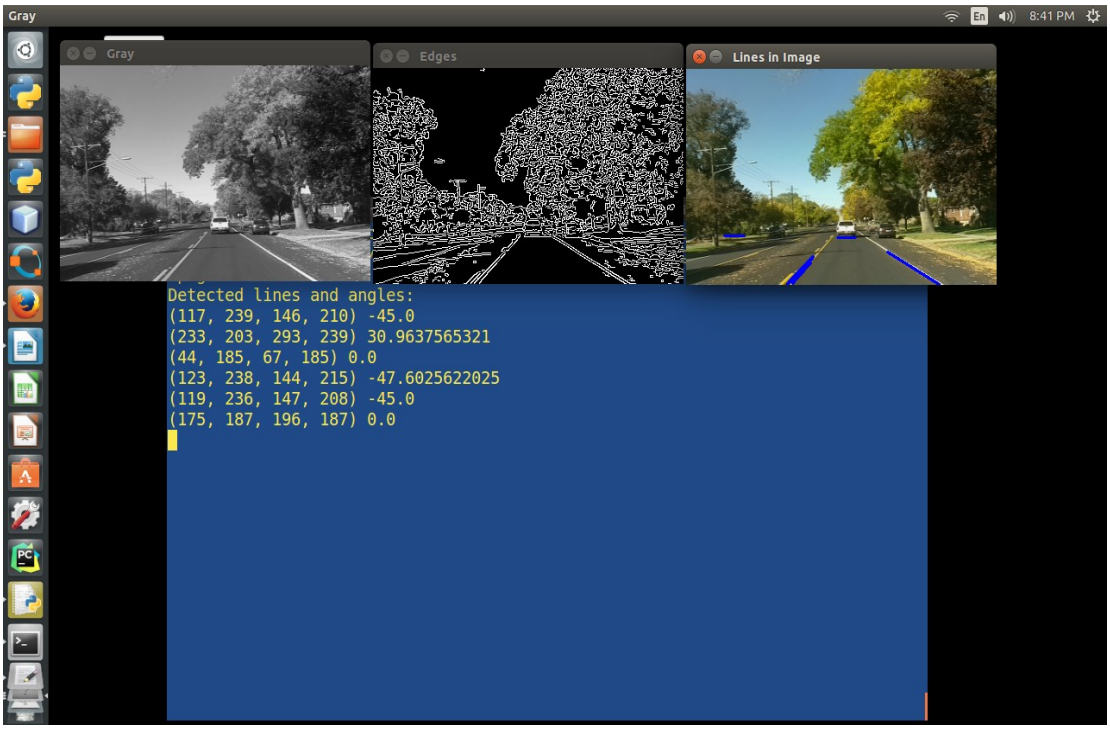

Figure 6. Result of calling display_ht_lanes on a image

As shown in **Figure 6**, all detected lines are displayed in the terminal in terms of their points and angles. I deliberately not show the command line values of **num_votes**, **min_len**, and **max_gap** that pass to this call on command line, because I want you to experiment with these values to discover which values give you optimal results.

2. Detecting Left and Right Lane Lines

Since the detected lines are never perfectly diagonal, we need to define a function **is_angle_in_range(a, ang_lower, ang_upper)** to test if an angle a is in a specific range **[ang_lower, ang_upper]**. This will allow us to define a diagonal line if its angle is in **[35, 45]**, for example. After you have defined **is_angle_in_range**, use it to define the predicates **is_left_lane_line(x1, y1, x2, y2, ang_lower, ang_upper)** and i**s_right_lane_line(x1, y1, x2, y2, ang_lower, ang_upper)**. You have to experiment with various values of **ang_lower** and **ang_upper**. After you have defined these predicates, use them in list comprehension to define two filters:

```
def filter_left_lane_lines(lines, ang_lower=<your value>, ang_upper=<your value>):
    pass

def filter_right_lane_lines(lines, ang_lower=<your value>, ang_upper=<your value>):
   pass
```

As their names suggest, these filters return a numpy array of lines and return a list of 4-element lists, i.e. lists of **[x1, y1, x2, y2]**, for the lines for which **is_left_lane_line** or **is_right_lane_line** return True.

The function **detect_ht_lanes_in_image** defined in **ht_detect_lanes.py** takes a path to an image and the arguments to **cv2.HoughLinesP()** and returns a 2-tuple where the first element is the number of left lane lines and the second one is the number of right lane lines detected in the image. It also generates an image where detected left lane lines are drawn as blue lines and detected right lane lines are draw as red lines. The image is saved in the same directory as the original image with **_lanes.png**. For example, a call to **detect_ht_lanes_in_image(/'road_images/16_07_02_14_27_16_orig.png', num_votes, min_len, max_gap)**, where **num_votes**, **min_len**, **max_gap** are some positive integers, will generate an image with drawn lines and save it in **'road_images/6_07_02_14_27_16_orig_lanes.png'** and will return a 2-tuple whose first element is the number of detected left lane lines and whose second element is the number of the detected right lane lines.

4. Detecting Lane Lines in All Images in a Directory

Define a generator **find_ll_rl_lanes_in_images_in_dir(imgdir, filepat, rho_accuracy, theta_accuracy, num_votes, min_len, max_gap)** that takes a Linux directory, a file pattern, and the arguments to **cv2.HoughLinesP()** and generates 3-tuples such that each 3-tuple consists of the file name, the number of left lane lines detected in it, and the number of right lane lines detected in it. Note that the value of rho_accuracy and theta_accuracy are set to 1 and numpy.pi/180, respectively. These values are typically used in **cv2.HoughLinesP()**, as we discussed in the lecture on Hough Transform.

```
def unit_test_04(imgdir, filepat, num_votes, min_len, max_gap):
   for fp, ll_rl in find_ll_rl_lanes_in_images_in_dir(imgdir, filepat, 1, numpy.pi/180, num_votes, min_len, max_gap):
```

print fp, ll_rl[0], ll_rl[1]

For example, a call **unit_test_04('road_images/', '16*.png', rho_acc, th_acc, num_votes, min_len, max_gap)**, for some specific values of **num_votes**, **min_len**, and **max_gap**, may output the following lines:

/road_images/16_07_02_08_24_59_orig.png 10 0
/road_images/16_07_02_08_25_35_orig.png 2 2
/road_images/16_07_02_08_25_48_orig.png 0 0
/road_images/16_07_02_08_25_49_orig.png 0 0
/road_images/16_07_02_08_25_50_orig.png 0 0
/road_images/16_07_02_08_25_05_orig.png 2 0
/road_images/16_07_02_08_25_09_orig.png 2 0
road_images/16_07_02_08_25_13_orig.png 6 0
/road_images/16_07_02_08_25_17_orig.png 4 0
/road_images/16_07_02_08_25_22_orig.png 6 0
/road_images/16_07_02_08_25_36_orig.png 4 1

5. What to Submit

Save your Py solution in **ht_detect_lanes.py** and submit in Canvas. State in your comments at the beginning of the file which values of ang_lower, ang_upper, num_votes, min_len, and max_gap worked best for you in detecting lane lines. Do you think it is possible to find universally optimal values for these parameters?

Happy Hacking!