CS 3630 Project 5

Team 1

Name: Narae Lee
GT username: nlee71
GTID: 903149574
Team #: 70

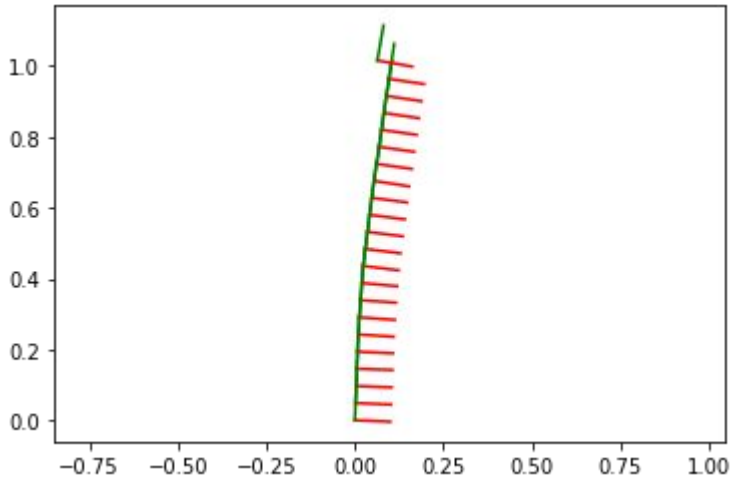
Name: Ziyang (John) Zhang
GT username: zzhang741
GTID: 90340370
Team #: 70

Team 2

Name: Andrew Wang
GT username: awang350
GTID: 903378837
Team #: 13

Name: Matthew Yang
GT username: myang349
GTID: 903447357
Team #: 13

1. Provide the screenshot of a plot showing the robot's orientation over time. Why might the final pose look slightly offset/out of place compared to the calculated trajectory (x, y, theta)?



The final pose is our actual measurement, and the other points are from our calculated trajectory. Thus, any error will be visualized as a jump at the very end of our plot. Explained more later, but the movement of our robot is not nearly as consistent as the calculated trajectory, and also, the camera is not a perfect tool in (pixel intensity noise, blurriness, background objects (such as another lane), etc.)

2. Explain the steps you used for detecting lane boundaries along the scan lines under “**find_lane_boundaries_on_scan_line**” function

find_lane_boundaries_on_scan_line () takes in an image, and a “scanline” v. Then, it finds the center of the left and right lane boundaries, at their intersection point with the horizontal scanline v.

First, we get the intensities corresponding to the given scanline. Next, we use np.convolve with the filter ([.1, .8, .1]) in order to make the data less susceptible to noise. With this filter, if a white pixel is surrounded by two white pixels (i.e. it is very likely to be part of our lane), it will still have a value of “white” after the filter. However, if we have a random white pixel surrounded by darker pixels, it will have a noticeably lower intensity value, which can exclude it from being included when we consider the pixels that form the lane boundaries.

Next, we iterate from the left edge to the center of the image, adding pixels with intensity >= 200 to left_list. Then we iterate from the center + 1 to the right edge of the image, again adding pixels with intensity >= 200. Lastly, we return the median for each list as u_left and u_right, the centers of our lane boundaries. Note: The intensity cutoff to be “part of our lane” is selected to be 200 in order to pass the unit test. In practice, on the TA_images, 240 would be a better cutoff.

3. Can your current lane boundary detection method also detect curved lane boundaries? Can you suggest some changes/methods for detecting curved lanes?

The current lane boundary detection method would do a poor job at detecting curved lane boundaries because it only uses two points to form a line. That is, our list of depth_values for scan lines is simply [50, 100].

From the collab file, step 5 finds pixels for lane boundaries. Step 6 uses these pixels to compute camera coordinates, then step 7 fits a line along these camera coordinates.

If we only use two depth_values for step 5, then the calculated lane boundary will be linear. Change: We can detect curved lines by computing the lane boundaries at many (not just 2) scanlines, and then we can fit a higher-dimensional curve to our computed coordinates, thus detecting curved lines.

4. Include a screenshot of your Unit Test results below:

5. What might be the reasons for error in final orientation of the robot when compared with the ground truth orientation angle that you recorded in Lab part 1?

Our results were:

- Computed theta in radians: -0.14510584111216626
- Computed theta in degrees: -8.313952278423034

In comparison, the ground truth orientation angle was -10 degrees, which is objectively very close, given the estimated “You can expect around 5-10 degrees of error in the overall calculation of theta.”

The biggest reason for this error is with the algorithm used in our code. Note that our code finds theta by “computing the angle of the optimal path”, then simulates the robot traveling total_dist / (num frames) in the direction of theta at each time step.

However, we only have 20 frames. Meaning, in the simulation, our robot will only change direction 20 times, and it will go along a straight path during each interval. **In reality, our robot is constantly curving, moving left, then right (resulting in changes in orientation), even in small windows of time despite a perfectly adjusted trim.** Thus, it is impossible for our code to perfectly capture the final orientation of the robot.

6. Specify the challenges faced during the lab

Biggest challenges were with Collab, and our photos. Despite the name, it is very difficult to work simultaneously on the same Collab file, as it constantly sends errors about saving edited code (even in unrelated areas), etc.

Also, even though our photos were checked off by the TA, the issue of having other lane boundaries (other than ours) in the photo completely messes up the lane boundaries algorithm. This is because when you go along the scanline, the intensity values will spike 3-4 times instead of the desired two times, since there are other lanes in our photo.

However, because the other lanes are thinner than ours, using the median “white” pixel still resulted in functioning code. **However, we decided to go and use the TA photos (after confirmation that this was allowed) because they were much clearer anyway.**