CPE470 Midterm Written Section

Justin Trinh

October 2025

1 Question 1

See polar_to_cartesian.py

Created a measurement class with attributes of the polar and Cartesian coordinates. When initialized with polar coordinates, the class automatically calculates the corresponding Cartesian coordinates with the following:

x = distance * cos(angle)

y = distance * sin(angle)

The read_file function reads all lines of the file and creates an array of Measurement object that stores all of the lidar data.

2 Question 2

See inflexion_points.py

Points E and F represent the endpoints of the walls. When we plot all the LiDAR measurements in polar coordinates (as in Figure 1), there is a gap in the data around 200 degrees, where the sensor did not detect any surfaces. This gap corresponds to an open space between two walls.

To identify this gap, the program iterates over all consecutive data points and computes the angular difference between each pair, where the largest angular gap indicates the break between walls. The last point before the gap and the first point after the gap are then the points E and F. In this dataset, these points are located at E = (107.71, 23.76) and F = (135.95, 120.85), with an angular separation of 29.19 degrees.

Points A, B, and C are points that mark corners in the walls. As seen in Figure 1, they appear as local maxima in the dataset, as in that general direction, the corners would be the farthest point away from the lidar scanner.

In theory, the corner points are the inflection points where the distance switches from increasing to decreasing. This means that the first derivative of distance with respect to angle is positive in the angle interval less than the inflection point, whereas the derivative is negative in the angle interval greater than. We can then detect this change in the derivative through the use of the second derivative of distance with respect to angle. The second derivative should have a large negative value, as the first derivative is suddenly going from positive to negative. The corresponding points that have the largest negative value are our corners of A, B, and C.

$$\frac{d(distance)}{d(angle)} = \frac{distance_{cur} - distance_{prev}}{angle_{cur} - angle_{prev}}$$

$$\frac{d^2(distance)}{d(angle)^2} = \frac{(derivative_{cur}) - (derivative_{prev})}{(angle_{cur}) - (angle_{prev})}$$

However, in practice, this method does not produce meaningful results due to the amount of noise in our data. In addition, because our dataset is extremely large, the angle differences used to calculate the first and second derivatives are extremely small, causing the noise to be extremely amplified in our calculations. To overcome this issue and reduce the noise in the calculations, I used a trimmed moving average for each data point in order to reduce the noise in the data and then sampled a smaller subset of that data set in order to mitigate the effects of the extremely small angle difference between points.

For the trimmed moving average, I used a window size of 21 and a trimming of 7 elements. This means that for each data point, the average value that would be used for that data point would take into consideration 21 data points: 10 right below, 10 right above, and the original data point. First, I took the median of the 21 values and then calculated the deviations of each data point from this mean. Then I would remove the 7 data points with the highest deviations from consideration. This trimming allows me to ignore the effects of outliers caused by the noise. After this, I would take the average of the remaining 24 data points and use that value as the new value for that data point. I only took a subset of this averaged data set by creating my average data set with only the calculations from every 10 data points.

Using this new data set with reduced noise and greater angle differences, I applied the first and second derivative processes described previously. As a result, the locations of points A, B, and C are (-120.88, 75.98), (71.03, 104.95), and (97.01, -89.52) respectively.

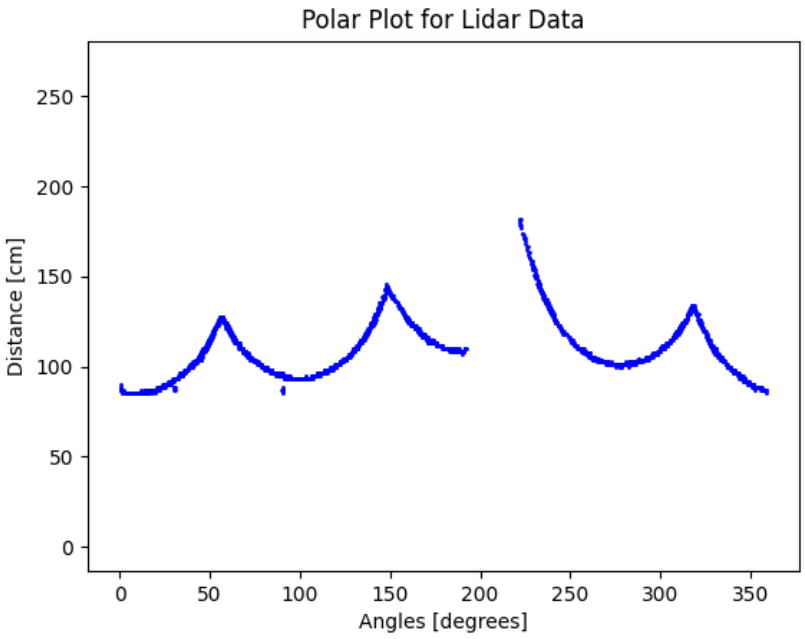

Figure 1: All data points are plotted with the x-axis being the angle and the y-axis being the distance.

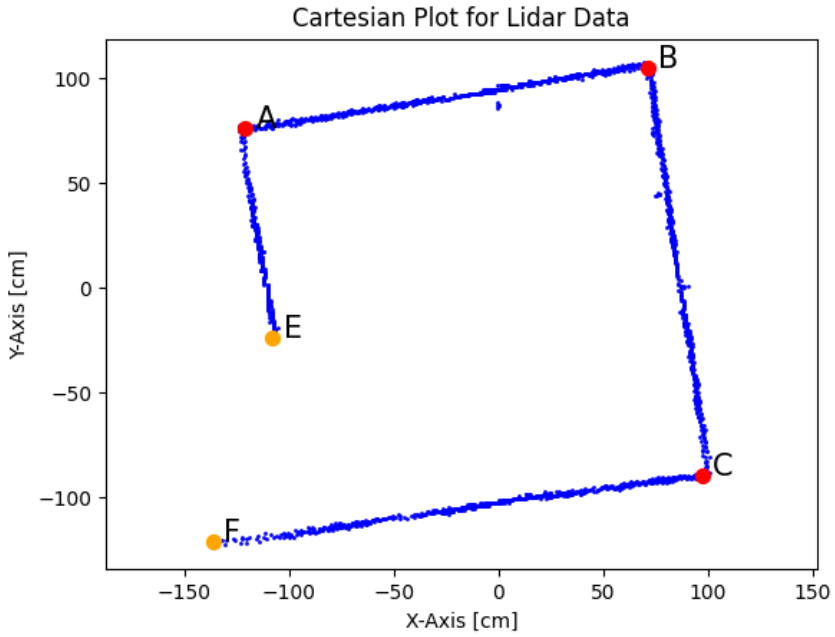

Figure 2: Final resulting Cartesian graph with all data points and calculated corner and wall end points.

3 Question 3

See least_squares.py

Finding the Equations of the Lines Representing the Walls
1) I split the data set into 4 sub-sets, each representing a wall. For each subset, I had two data points representing them, the two corner points, and I only added data points that had an angle within the range formed by the angle of the two corner points.
2) To get an estimation of the equation of the line, I will use the weighted least squares algorithm to calculate m, the slope, and c, the y-intercept, of the line. For the weighted least square equation, our X = [[m], [c]] and the Jacobian matrix of H has to be [x, 1] for the equation y = HX + v to be y = mx + c as the equation of the line. To create the Jacobian matrix we will simply add a row of [xi, 1] for every x coordinate measurement we have. For the y vector, it is simply a vector containing all the y coordinate measurements. The resulting x_hat is a vector with [[m], [c]] as the weighted least square estimates of the m = slope and c = y-intercept of the line. This process is then repeated for all 4 of the subsets.

$$\hat{X} = (H^T H)^{-1} H^T y$$

y = mx + c

Finding Corner D
The corner D is the intersection of where the wall of points A to E and the wall of points F to C would be if there were no gap in the wall; therefore, we can find point D by finding the intersections of the line that represents the wall of points A to E and the wall of points F to C.

We have the equation of both lines:
$y = m_1 x + c_1, y = m_2 x + c_2$
We set them equal to each other:
$m_1 x + c_1 = m_2 x + c_2$

We solve for x:
$x = (c_2 - c_1)/(m_1 - m_2)$

We solve for y using x:
$y = m_1 x + c_1$

The resulting (x,y) coordinate is the coordinate of the point D

Figure 3 shows the resulting plot with all of 4 line equations of the wall, all corner points, and point D overlaying the original lidar data points

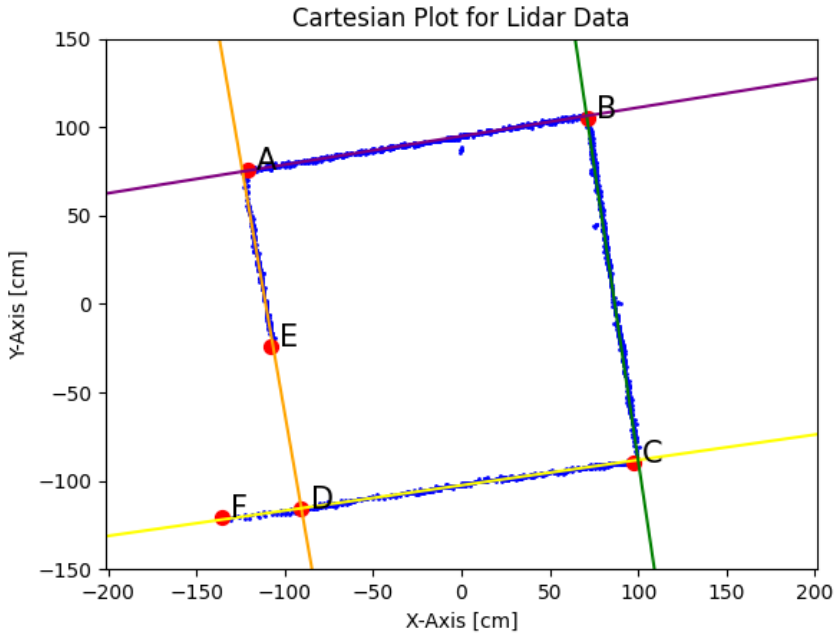

Figure 3: Plot of all 4 linear equations representing the walls and points A, B, C, D, E, and F

4 Question 4

See least_squares.py
find_wall_opening function and calculate_distance function

The wall opening in the room can be identified by finding the two consecutive points with the largest distance separation in between them. This seperaation is the gap in the wall and the corners are the two points that resulted in largest seperation.
To calculate the distance between two points I used the distance formula for cartesian coordinates with the current point for x1 and y1 and the previous point for point E for x2 and y2. I then repeated this process for all of the points in the data set.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Afterwards I found the maximum distance between two consecutive data points and set the point that had the maximum distance to be D and the previous point as E. These two points are the corners of the wall opening because they were the two points that resulted in the largest separation distance.
To find the width of the wall opening, I simply used the distance formula with the two corners of the wall opening resulting in a width of 101.11 cm.