Checkpoint 1.3 Progress Report

Logan Zartman

Austin Atchley

1. Mathematical computation:

LIDAR to point cloud computation:

We used the calculations presented in class.

for each point in point cloud: theta_i = theta_0 + i * theta_increment p_i = {r_i * cos(theta_i), r_i * sin(theta_i)}

Obstacle detection computation:

1. Given command line flags for max distance and curvature
2. Assuming we already have the point cloud constructed
3. Free path length = max distance
4. Iterating through all points in point cloud:
    1. If point is not an obstacle (using formula presented in slides), skip.
    2. Find minimum of free path length with this point, and the free path length so far
5. Add free path length to our current position to get target position
6. Run 1-D TOC with our current position and this target position

2. Code organization:

We refactored our 1D TOC to take only distance and curvature as parameters. It exists entirely as a function that operates on these parameters and the state. Broadly, the functionality we added for this checkpoint determines the distance and curvature necessary to run 1D TOC correctly.

We have a function called `_is_in_path` that returns a boolean value. It performs the calculations outlined above in section 1.2. If the point is in our path, we compute the distance along our path to the point with `_get_dist_to_point`. This is the maximum distance we can go. If the object moves, the value for `_get_dist_to_point` is updated accordingly, and the car resumes its previous path.

Also, if the curvature is close to 0, our code branches into a straight version of `_is_in_path` which performs a different calculation to avoid floating point error.

3. Parameter tuning:

We estimated and measured physical dimensions on the car. Otherwise, the parameters we selected in previous checkpoints held up well in our testing.

4. Challenges faced:

- We had trouble debugging our obstacle detection code because we could not make visualizations work. Once we were able to display our point cloud, we were able to pinpoint the issue specifically to the obstacle determination code.
- We programmed our obstacle-in-path check incorrectly at first. We debugged this by visualizing all points considered to be within the path of the robot. After fixing a mathematical error, we could visually confirm that the obstacle checking worked correctly.
- We defined the size of our robot incorrectly. We noticed this by visualizing the bounding box of the robot.
- We forgot to transform from the LIDAR frame to the base link frame. We debugged this by noticing that the robot stopped short of obstacles.

5. Contributions of each team member:

Austin - Point cloud reconstruction and visualization. Refactoring and structure of obstacle detection.

Logan - Mathematical computations. Fine-tuning of obstacle detection.

All object detection code was pair-programmed with Austin committing, but both of us shared the load about equally.

6. GitHub Repo

https://github.com/austinatchley/F1-10-Autonomous-Driving

7. Video Demos

Low positive curvature value

https://drive.google.com/open?id=1pZqY6qFQXsJXYHtotJn4z4DaEXDspow8

High negative curvature value

https://drive.google.com/open?id=1sn9RkY4Kmy1uIlK_Elz_0V7_eoUz6u3L

Low positive curvature value with moving obstacles

https://drive.google.com/open?id=1LvRrm8nX1ExARm2PeNV4seEy5ZIirfrO

8. Future improvements:

- We would like to reduce twitching when the car stops.