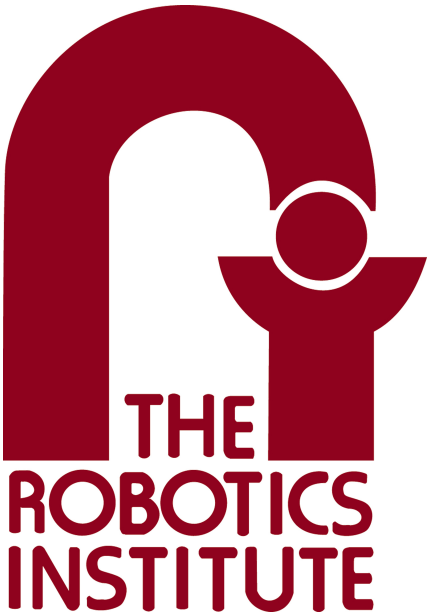

MRSD Project Course

Team I – AIce

Autonomous Zamboni Convoy

Individual Lab Report 3

Team

Rathin Shah

Nick Carcione

Yilin Cai

Jiayi Qiu

Kelvin Shen

Author

Kelvin Shen

Mar 3, 2022

Table of Contents

Individual Progress 2

Challenges 5

Teamwork 6

Plans 6

Individual Progress

During the past few weeks, I focused on finalizing the perception module that will be run on a Zamboni convoy where both vehicles will be moving instead of in a setting with only one Zamboni and a static wall of ArUco markers (as shown in the last ILR). In addition, I calibrated the RealSense D435i in reality, tested and calibrated my pose estimation algorithm on it using a printed marker board.

Pose Estimation Algorithm Test with two teleoperated Zamboni's

To test the perception module in a setting closest to reality, we need to attach the marker board to the back of another Zamboni (leader) and estimate pose as well as depth from it. In the last progress review, we only showed the working perception module given a teleoperated Zamboni along with a static wall of a marker board.

However, instead of attaching the same board used in the last progress review to the rear of the leader Zamboni, I remade the URDF of the marker board because of the inconsistent scale issue (further explained in *Challenges*). In particular, I added the board image as a texture into the Gazebo materials directory so that we can directly render a plain box using the material tag in its URDF. After solving the scale issue by remaking the marker board, I attached it to the rear of the leader Zamboni according to the dimensions of the Zamboni URDF file. Figure 1 shows the current setting of our simulation.

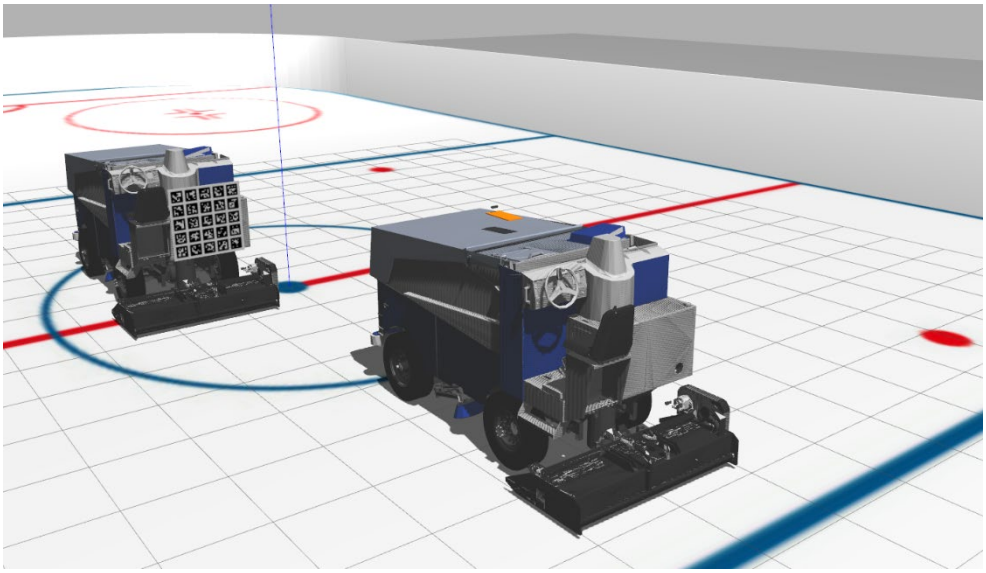

Figure 1. Leader (with marker board) and Follower (with D435i) are Launched in Gazebo

To validate the output from the pose estimation algorithm, I retrieved ground truth poses of links in Gazebo. In particular, thanks to Yilin's contribution which makes it possible to have both tf trees of the two Zamboni in Rviz, I looked up the transform

using tf to find the transform from the camera link to the marker board link, of which the norm of the translation vector gives the ground truth depth while the quaternion gives the ground truth rotation.

Finally, I tested my pose estimation algorithm by comparing it with the ground truth. I spawned two Zamboni's with both a longitudinal and a lateral offset, teleoperated them with a constant speed, and kept estimating the pose of the marker board. The output of the algorithm contains three entities: one is the transform from the marker board frame to the camera frame, one is the norm of the translation vector (which is equivalent to the distance from the camera to the origin of the marker board frame), and one is the interpolated average depth. The interpolated average depth is calculated as follows: (1) when there are at least four markers detected on the marker board, pick out four of them that are as far from each other as possible, which means ideally the four markers picked out should be on the four corners of the board; (2) generate a binary mask that is a contour using the center positions of these four markers; (3) interpolate the depth image (which must be already aligned with the RGB image) using the binary mask and calculate the average of the interpolated depth image to get the depth estimate of the board. The reasoning behind the interpolation is that ideally I want the depth estimate to be based on the distance from the camera to the center of the board, which is the origin of the board link defined in URDF. However, there must be situations when the center of the board is occluded so we cannot simply hardcode the center of the board to be the position used to retrieve depth in the depth image. We need to adaptively change the region we use to retrieve depth values from the depth image based on the marker detections we got from the RGB image.

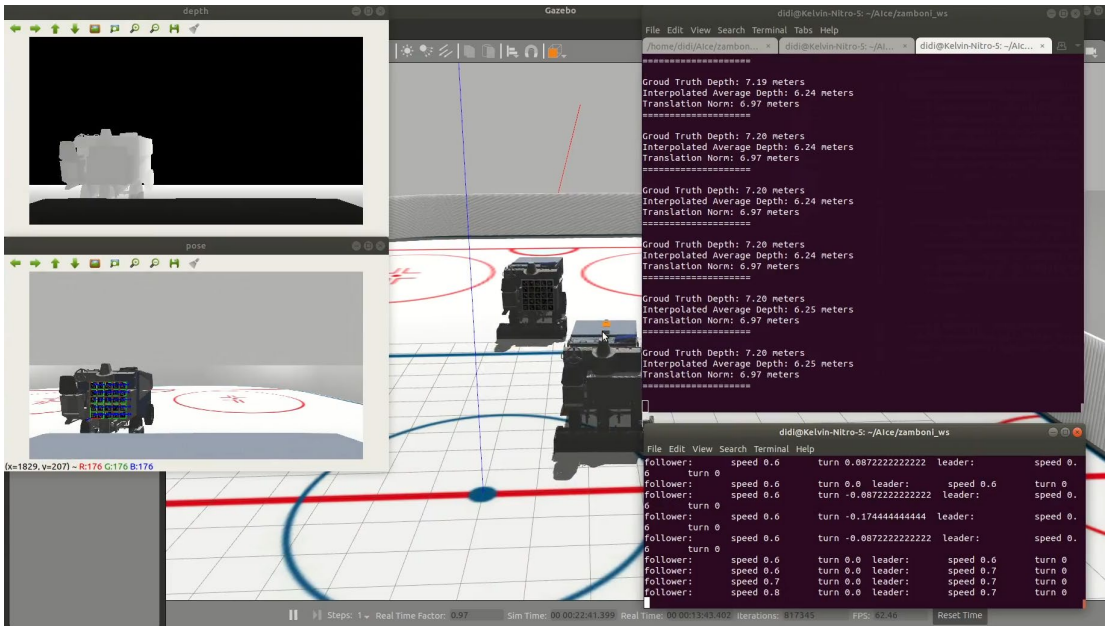

Figure 2. Pose Estimation of the Leader in Gazebo

Figure 2 shows the output of the pose estimation algorithm. In the terminal, there are three outputs per image callback, “Ground Truth Depth” is the norm of the translation vector of the transform we looked up using tf. “Interpolated Average Depth” is the estimated depth based on the interpolated depth image as explained above. “Translation Norm” is the estimated depth directly based on the translation vector from the pose estimation of the board. Ideally, these three values should be the same. In practice, I found sometimes the interpolated depth images have outliers that lower the average depth, and I am planning to try calculating the median instead. The translation norm based on the pose estimation of the board in RGB image gives the closer result to the ground truth. The little difference between the two is caused by different origin of the board frame used. In calculating ground truth, tf takes the center of the board as the origin of the board frame, as defined by the URDF, while OpenCV takes the bottom left corner of the board image as the origin of the board frame as its convention.

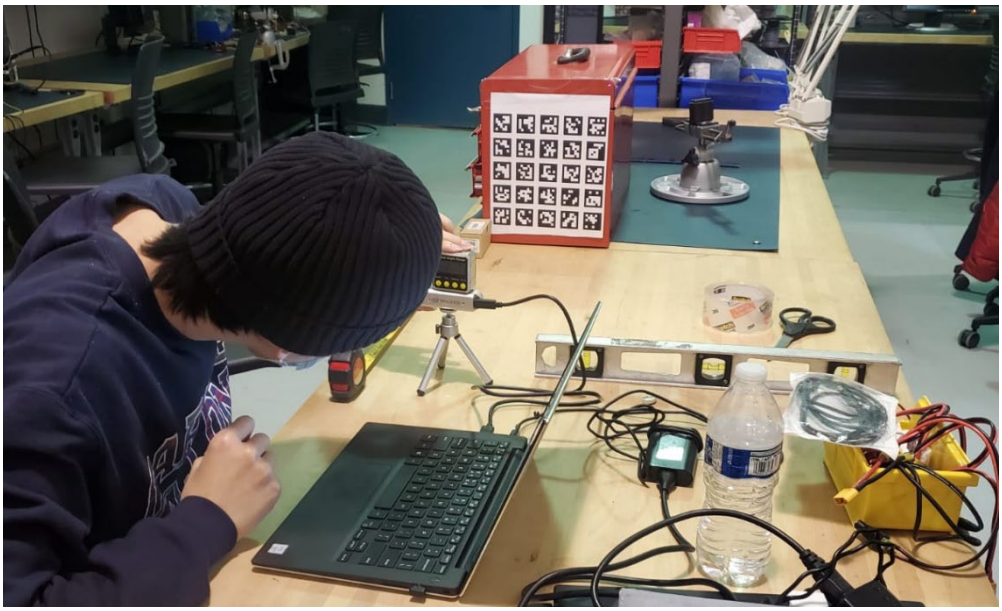

Figure 3. Marker Board Pose Estimate Validation using D435i

D435i Calibration and Validation

I calibrated Intel RealSense D435i by following the calibration guidelines using realsense-viewer. I tested my pose estimation algorithm by first validating the norm of translation output, which is equivalent to depth, against a tape measure between the camera and the printed marker board. Figure 3 shows the setting for the validation. The printed marker board is the same board image used in the simulation above. Then I validated the rotation output by (1) attaching the marker board to a hinge that can rotate 90°, e.g. the red toolbox with a lid in Figure 3; (2) calculating the rotation

difference between the current rotation matrix and the rotation matrix after rotating the marker board by 90° around the axis of the hinge; (3) converting the rotation difference into axis angle and verify the angle was indeed 90°. In this way, we successfully avoided any artificial errors such as the marker board is not perfectly parallel with nor in the center of the image frame, because what the axis angle gives is always around the axis of rotation. Finally I tested and validated the depth camera by interpolating the depth image using individual marker detections' locations, same as in simulation.

Challenges

The major challenges I have encountered when implementing the tasks above include:

1. Getting ground truth position of links in Gazebo
   Rather than hardcoding the coordinate or any transform, we wish to neatly obtain the information of interest using functionalities provided by Gazebo or ROS. Initially we tried to listen to the topic "gazebo/link_states" which publishes pose messages of each link in Gazebo. However, Gazebo lumped all fixed joints in a sequence (and hence links connected by them) to be a single joint, which makes impossible looking up the index of the camera link by subscribing to the topic. I solved this by looking up tf transforms between the camera link and the base link, and multiplying the transform by the ground truth position of the lumped base link that is able to be indexed in messages from "gazebo/link_states" topic. Similarly I used this approach to get the ground truth position of the marker board link.

2. Marker board wall scaling issue
   Originally our marker board model in Gazebo was auto-generated using an open-source script. The board seemed to work at first but when we take the norm of the translation vector from the pose estimate, the scale was inconsistent with how we generated the board image. In particular, our board image is generated to be 390 by 390 pixels, and the side length of the box model, to which the board image is attached, is set to be 1 meter in Gazebo. Therefore, if we divide the norm of the translation vector, which is in pixel, by 390, we should get the depth value in meter, but this was not the case using our original marker board model. My solution was to remake the marker board URDF by making the board image as a custom Gazebo texture so that we can directly render a plain box using the material tag in its URDF definition.

3. Texture can't be loaded in Gazebo by any means
   This happened when I changed the definition of a Gazebo texture in an existing texture file. When I relaunched any model using that updated texture, it wouldn't reflect in Gazebo. I haven't found any solution to this so far. I have verified that my gazebo_media_path environment variable includes the path to the updated texture file. The current solution was to make any change to the launch file and it will "force" Gazebo to rebuild its resources folder and be able to loop up the updated texture.

Teamwork

- Nick worked on the RC car platform, including figuring out how to steer the wheels and send speed commands to motors, as well as making possible controlling the RC car by either manual input or computer command.
- Rathin worked on the path following module in Simulink. He designed and 3D printed necessary mounts for the sensors on the RC car. He also mapped the steering angles on servo to the real steering angles for the RC car.
- Jiayi worked on the leader velocity estimation algorithm using EKF based on its position and heading angle relative to the leader. She also designed and published waypoints including positions, heading angles, and velocities for the vehicle controller tests as well as PID tuning.
- Yilin worked on the dynamics parameters of the Zamboni URDF so that they were as close to a real Zamboni as possible. He tuned PID parameters in ROS Controller to smooth the teleoperation on the Zamboni in Gazebo. He also integrated waypoints loading and pure pursuit controller, Rathin's work, to realize path following in Gazebo simulation along with Rviz visualization.

Plans

Before next progress review, I will help integrate the leader velocity estimation algorithm, developed by Jiayi, with my leader pose estimation part. After that, we would be able to integrate all subsystems and demonstrate a basic leader follower system in simulation. I will also work on leader detection using D435i on the RC car in a dynamic environment.