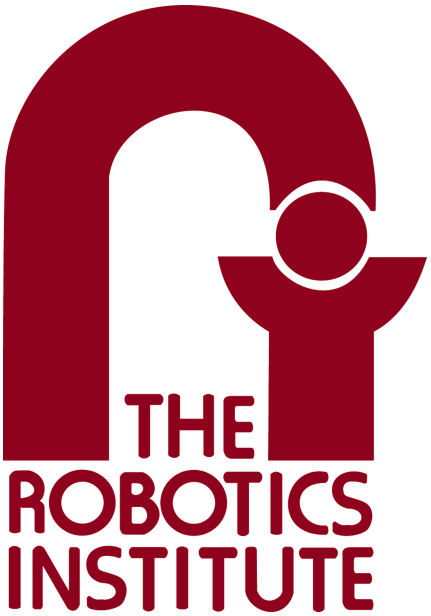

MRSD Project Course

Team I – AIce

Autonomous Zamboni Convoy

Individual Lab Report 2

Team

Rathin Shah

Nick Carcione

Yilin Cai

Jiayi Qiu

Kelvin Shen

Author

Kelvin Shen

Feb 16, 2022

Table of Contents

Individual Progress

Over the past week since my last individual report, I focused on setting up and making sure the fundamental elements of our perception stack are working in the simulation.

Integration of RealSense D435i into Gazebo

I integrated and imported a RealSense D435i camera into our Gazebo environment. Specifically, I installed it on our Zamboni model as shown in Figure 1. The Gazebo plugin comes with the flag that can visualize what the camera is seeing in real time in Gazebo. The RealSense does not have an official package that launches its cameras into Gazebo; it only provides a ROS wrapper that interacts with the physical RealSense camera, instead of in simulation. Therefore, I integrated separate packages, realsense2_description which provides 3D-models of the camera, and realsense_gazebo_plugin which enables simulation of the RealSense D435 cameras in Gazebo.

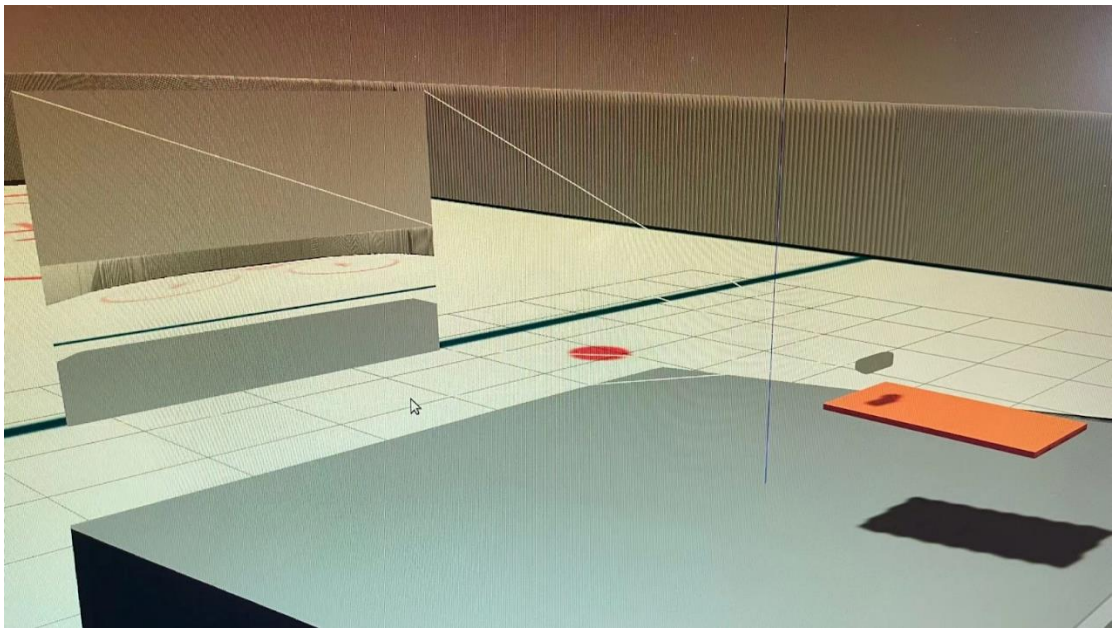

Figure 1. Zamboni Model with D435i Launched in Gazebo

Pose Estimation using ArUco Board and RGB Image

With the RGB-D camera set up, I spawned a wall of ArUco markers inside Gazebo for the pose estimation of the leader Zamboni. I chose an ArUco board for detection over several individual ArUco markers and the reason is twofold: (1) the pose estimation on a board becomes much more versatile because the arrangement of a board is fixed once I initialize it using a specific dictionary of markers and hence only some markers are necessary to estimate the pose (the algorithm can deduce the other

blocked or blurred markers based on the ones that can be clearly seen); (2) the obtained pose would be much more accurate due to a higher amount of point correspondences (marker corners) compared to individual markers. For example, as shown in Figure 2, the algorithm only detects 12 markers correctly (the one in the third row and the first column is a postprocessed, refined one using the detected markers locations), but it still gives an accurate pose estimation, i.e. the xyz axis drawn on the top right corner of the board. The pose estimation algorithm works as follows: it first detects individual marker in the camera image, extracts the locations of each detected marker's corners, (optional) infers those undetected markers nearby using the detected corners, and finally estimates the rotation and translation vector from the board frame to the camera frame based on the given camera intrinsics and distortion coefficients. The norm of the translation vector can actually tell how far the camera is from the top right corner of the board, while the rotation vector (in axis angle) can tell the pose of the leader Zamboni relative to the follower as long as the ArUco board is attached to the rear of the leader.

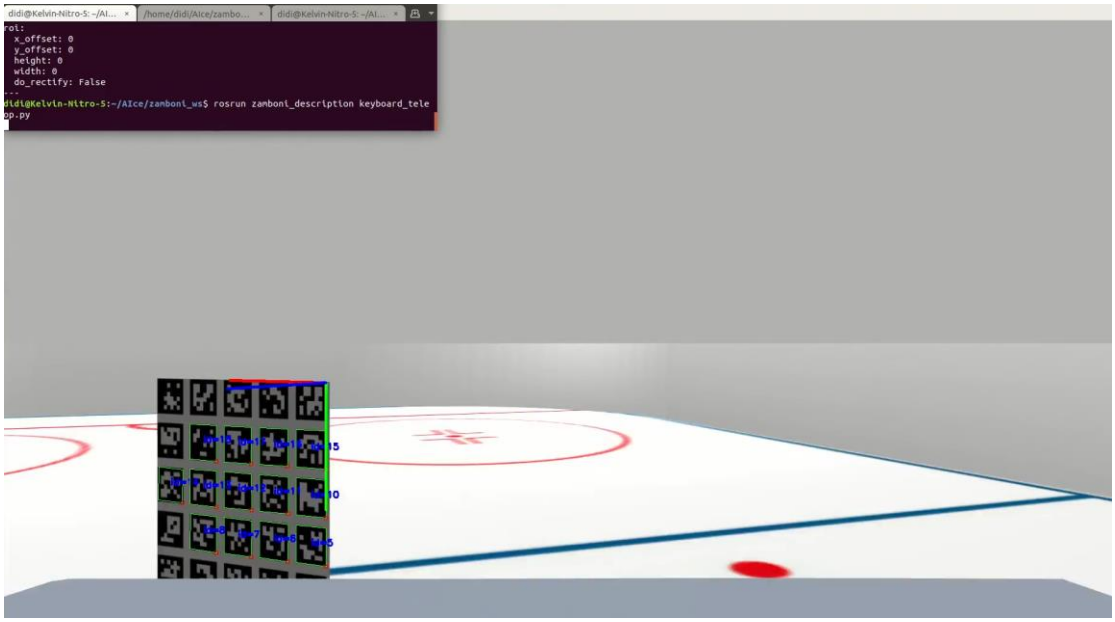

Figure 2. Pose Estimation of ArUco Board using D435i

Depth Information Extraction from Depth Image

Although the translation vector tells the distance from the follower to the leader, we still want to know the value obtained from the depth camera on D435i in case the pose estimation doesn't work in edge cases such as all markers are blocked or the board is in parallel with and right in the center of the camera image. To do this, I cropped out a region of interest (ROI) by picking out four "corner" detected markers that can expand the largest area over the board. I implemented the ROI cropping in this way because we not only want to use the region that are robustly detected based on the markers, but also want the region used to extract depth values as large as possible so

that noise in the depth image can be averaged out. As illustrated in Figure 3, I picked the top left, top right, bottom left, and bottom right markers that are correctly detected and used them to crop out an ROI. The ROI can then be passed as a mask on the depth image to get the average depth inside that ROI. The complete pipeline is shown in Figure 4.

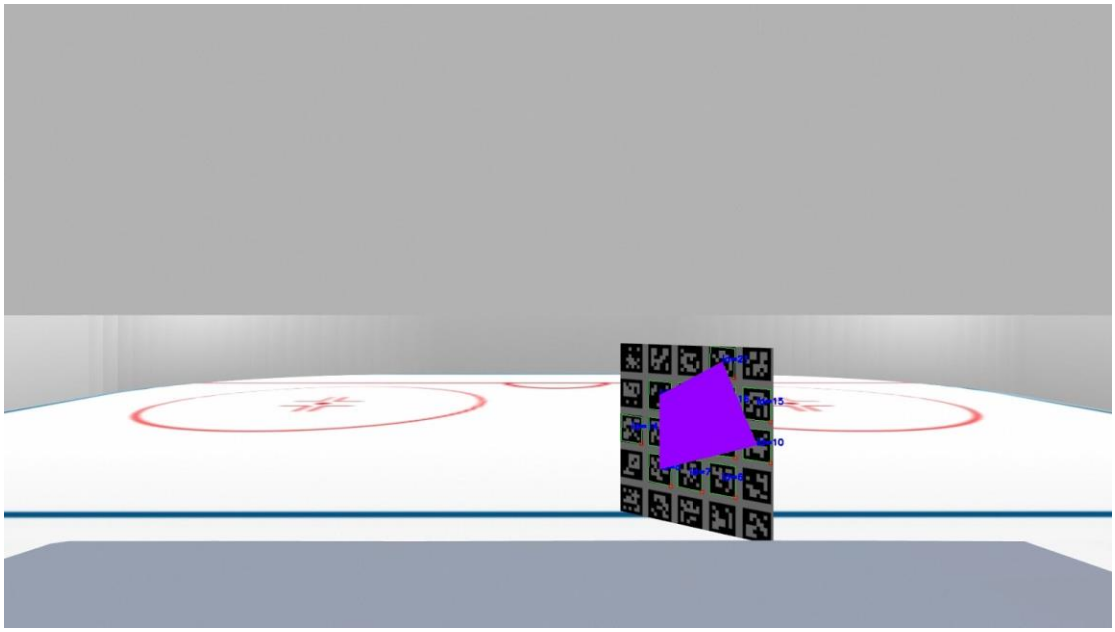

Figure 3. ROI used to extract depth from depth image

In the complete pipeline, the depth image and the RGB image has been aligned with the same FoV. They were not aligned originally due to different specs of the cameras on D435i (depth camera has an FoV of 85° and RGB camera has an FoV of 69°). I hardcoded the FoV of the depth camera to be 69° inside the Gazebo plugin definition.

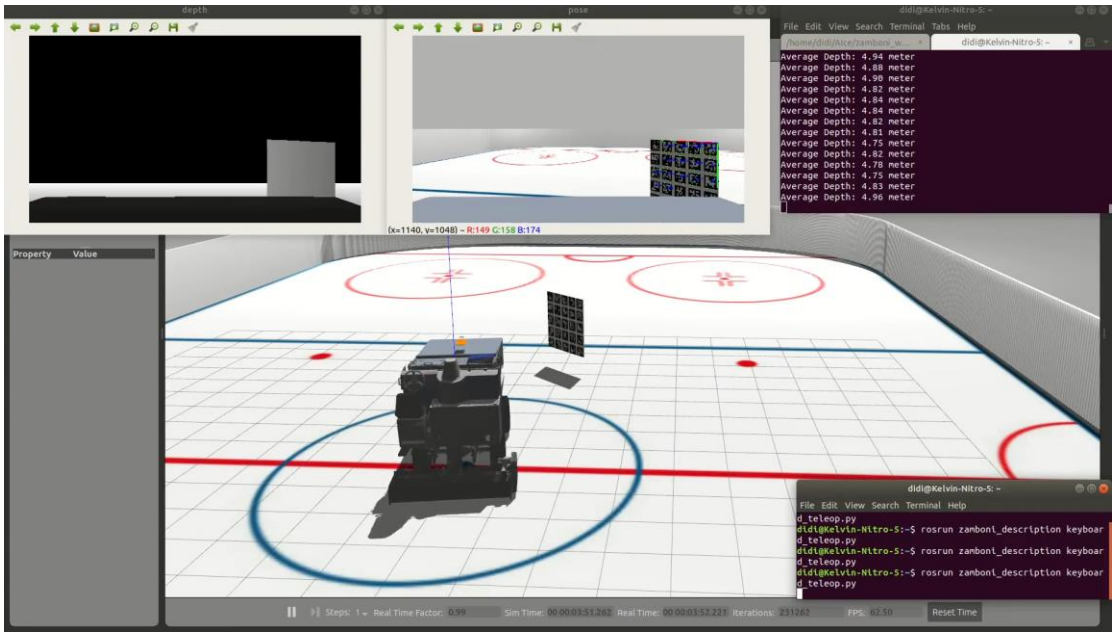

Figure 4. Complete Pipeline

Challenges

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

- The default resolution of the RGB camera in the Gazebo plugin implementation was 640x480, which resulted in poor performance of marker detections. I dug into the realsense_gazebo_plugin, from the highest level of the launch file to the lowest level of the xacro file for the basic D435 camera, and hardcoded the resolution to be 1920x1080, which is consistent with the specs in D435i documentation.
- The camera intrinsics and distortion coefficients are required when estimating the pose of the board. They are published on the camera_info topic but I don't wish to subscribe to the topic every time stamp with callbacks because I only need these parameters once and they would be unchanged for the rest of any program. I accomplished this by learning to use ros::topic::waitForMessage() that enables me to echo the message on a topic only once without a callback.
- We wish to have one node that publishes both the pose estimation and the depth per our subsystem requirement. The pose estimation is retrieved using the RGB image while the depth is retrieved using the depth image. However, the RGB image and the depth image are published on two different topics, which make it redundant to have two callbacks in one node. Plus, since I need to apply the ROI cropped from the RGB image as a mask to the depth image, it'd be optimal to have access to both images under the same scope instead of using global variables to communicate. I accomplished this by learning to use message_filter package; specifically, its time synchronizer allowed me to subscribe to both topics and to synchronize them by the timestamps contained in their headers, in which way we are able to use a single callback to process messages from multiple channels.
- The pose estimation result returned from the cv2::aruco library is ambiguous. I looked up OpenCV and ArUco documentation to understand the transform represented by the pose returned. The result was the transform from the frame of the board to the camera frame, and this is indeed the information we need for localization. Combining this with the transform we got from the follower Zamboni localization (the transform from the world frame to the odometry frame), we can now chain the transforms and get the estimation of the leader's pose in the world frame.
- The depth image and the RGB image are not aligned initially. The Gazebo plugin for D435i doesn't provide an aligned_depth_with_color topic that aligns the depth image with the RGB image (which was provided in RealSense

official ROS package for the physical camera). The RGB image has an FoV of 69 degrees and a resolution of 1920x1080, while the depth image has an FoV of 84 degrees and a resolution of 1280x720. I managed to align their FoV by hardcoding the Gazebo plugin implementation for the depth camera without losing generality. However, I can't hardcode the resolution in a similar way because that would be against the physical specs of these two cameras.

Teamwork

- Nick worked on the hardware components necessary for drive-by-wire conversion and began looking into their availability. He investigated the availability of the hardware platforms that we could use to demonstrate our autonomy stack. He also worked with Rathin on the program management presentation.
- Rathin worked on the localization algorithm with Yilin and integrated his pure pursuit controller developed in Simulink into Gazebo. He also worked on the schematic for the PCB assignment.
- Jiayi worked on the ROS navigation package as well as the move_base package in preparation for the trajectory generation and planning after localization and perception are finished.
- Yilin continued to work on having two Zambonis independently controlled by keyboard in a single Gazebo environment. He also fused odometry and IMU data using robot localization package in order to get the follower's pose.

Plans

I would keep on working on the perception stack in ROS. First I have to solve the inconsistent resolution in the RGB camera and the depth camera by resizing my ROI based on the corresponding aspect ratio. Then I need to compare the depth values extracted, as well as the norm of the translation vector, with the ground truth distance between the camera and the board using some Gazebo built-in functionality. And finally I will publish the depth and the transform (from the board frame to the camera frame) onto a node, which would be used for trajectory generation.