MRSD Project Course

Team I – AIce

Autonomous Zamboni Convoy

Individual Lab Report 5

Team

Rathin Shah

Nick Carcione

Yilin Cai

Jiayi Qiu

Kelvin Shen

Author

Kelvin Shen

Apr 6, 2022

Table of Contents

Individual Progress

During the past few weeks, I focused on solving the problem of losing track of markers when the leader Zamboni ice resurfacer makes a turn. In simulation, I solved the problem by setting up two cameras on the follower and three marker boards on the leader. In addition, I validated this new configuration with the waypoint generation and redesigned the perception pipeline accordingly. In parallel, I set up the environment for the autonomy as well as the interface between ROS and Arduino on Jetson AGX Xavier. I also managed to run the Husky software and control it using a gamepad.

Two Camera Setup

To avoid losing track of markers when the leader Zamboni ice resurfacer is making turns, I added one more RealSense D435i to the follower. This, along with the original D435i, composes a $136°$ field of view, which is illustrated in Figure 1. I also added two more ArUco marker boards to the leader, one on the left and the other on the right, also illustrated in Figure 1. All the three marker boards on the leader are from three different ArUco dictionaries so that the detector can distinguish between them when iterating over these three known dictionaries. The number of markers is used to determine which marker board from which camera's view to be used as the final pose estimation, rather than randomly using one or fusing multiple detections. This is important because the goal of the pipeline is to find the transform from the leader base to the camera, and different boards from different cameras' views will have different transform, leading to at most 6 possible combinations. Therefore, either

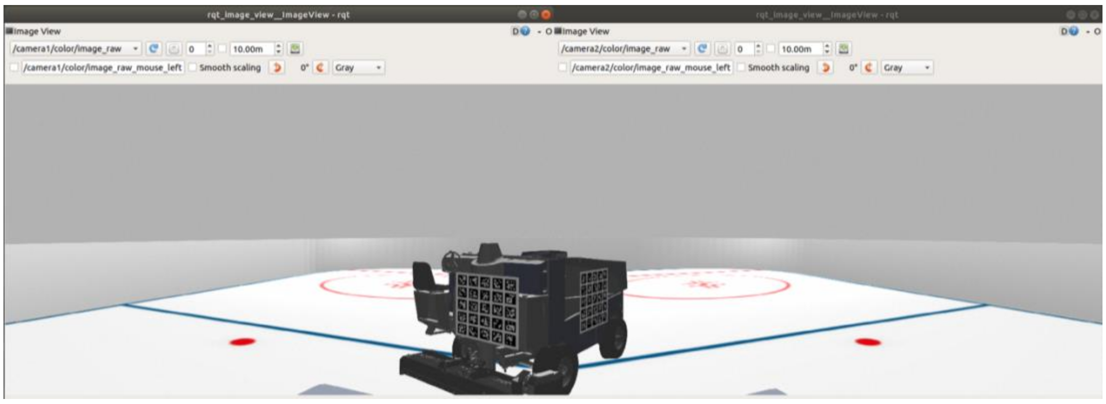
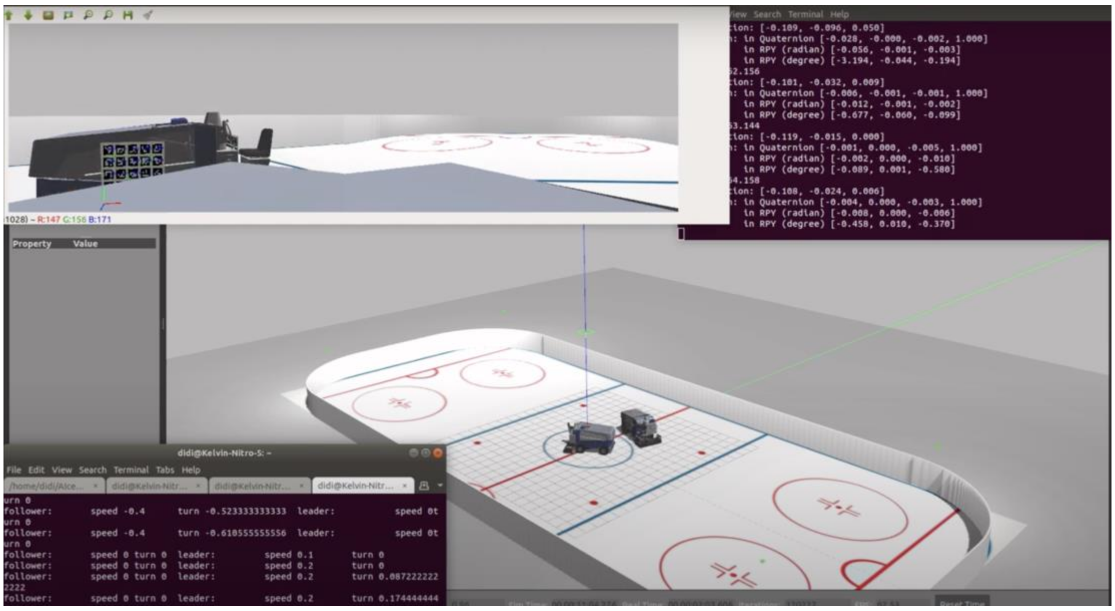

rqt_image_view__ImageView - rqt
Image View
/camera1/color/image_raw
0
10.00m
/camera1/color/image_raw_mouse_left
Smooth scaling
0°
Gray
rqt_image_view__ImageView - rqt
Image View
/camera2/color/image_raw
0
10.00m
/camera2/color/image_raw_mouse_left
Smooth scaling
0°
Gray

Figure 1. Two-Camera FoV

fusion or randomly picking a combination will not give the exact transform from the board (that is used to estimate the pose) to the camera (that detects the board). At the end of the pipeline, it transforms the estimated pose from OpenCV conventions to ROS tf conventions, as explained in ILR3, right-multiplies it by the transform from the leader base to the board (that is used to estimate the pose) and broadcasts the transform from the leader base to the camera (that detects the board). Figure 2 shows the result of our new perception configuration. The top left image concatenates two views from the two cameras and draws the estimated pose as an axis on the marker board. The top right terminal is calculating the rotation and translation difference between the estimated leader base and the ground truth. We can observe that even when the leader is almost making a U-turn, the translation error is still kept on a scale of centimeter per axis and the rotation error is kept within $0.5°$ per axis.



```
tion: [-0.109, -0.096, 0.050]
n: in Quaternion [-0.028, -0.000, -0.002, 1.000]
   in RPY (radian) [-0.056, -0.001, -0.003]
   in RPY (degree) [-3.194, -0.044, -0.194]
52.156
tion: [-0.101, -0.032, 0.009]
n: in Quaternion [-0.006, -0.001, -0.001, 1.000]
   in RPY (radian) [-0.012, -0.001, -0.002]
   in RPY (degree) [-0.677, -0.060, -0.099]
53.144
tion: [-0.119, -0.015, 0.000]
n: in Quaternion [-0.001, 0.000, -0.005, 1.000]
   in RPY (radian) [-0.002, 0.000, -0.010]
   in RPY (degree) [-0.089, 0.001, -0.580]
54.158
tion: [-0.108, -0.024, 0.006]
n: in Quaternion [-0.004, 0.000, -0.003, 1.000]
   in RPY (radian) [-0.008, 0.000, -0.006]
   in RPY (degree) [-0.458, 0.010, -0.370]
```

```
urn 0
follower:       speed -0.4      turn -0.523333333333  leader:           speed 0t
urn 0
follower:       speed -0.4      turn -0.610555555556  leader:           speed 0t
urn 0
follower:       speed 0 turn 0  leader:         speed 0.1       turn 0
follower:       speed 0 turn 0  leader:         speed 0.2       turn 0
follower:       speed 0 turn 0  leader:         speed 0.2       turn 0.087222222
2222
follower:       speed 0 turn 0  leader:         speed 0.2       turn 0.174444444
```

Figure 2. Pose Estimation Result when Leader is Making a U-turn

Validation with Waypoint Generation

The most dependent module on perception is waypoint generation. Only with a robust, consistent estimate of the leader pose will the follower follow a smoothly generated trajectory. However, during the integration of perception with waypoint generation, we have found that the delay of broadcasting estimated leader base leads to zigzag patterns in waypoints, as shown in **Error! Reference source not found.** (waypoints directly sampled from leader base position is in green, and the red waypoints are generated by adding the lateral offset to each leader waypoint, which will be followed by the follower). In particular, the tf broadcasted by the pose estimation script converges to the ground truth when the leader is static. However, when the leader starts moving, the estimated leader pose is always one timestep behind the ground truth, leading to the oscillating waypoints. To solve this, I optimized the script by removing redundant callbacks and exiting the loop that iterates over three boards and two

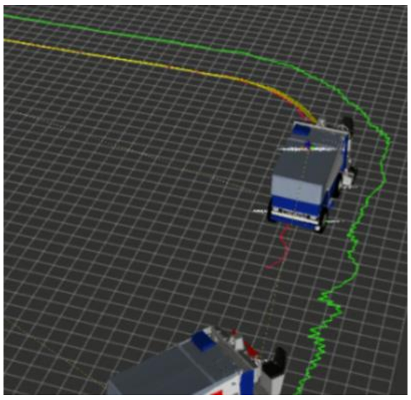


Figure 3. Zigzagging Waypoints

cameras whenever we can decide the optimal combination to be used for pose estimation. This improves the waypoints when the leader is moving straight ahead but the problem still exists when the leader is taking turns. Further smoothing is required for local waypoints before the lateral offset can be applied to them to generate follower's waypoints.

Back-up Platforms

I managed to set up the software for the Husky UGV and was able to remotely control it with a gamepad. Husky will be used as the back-up leader with a single ArUco marker board mounted on it. On the other hand, regarding the back-up follower, I have set up the environment on Jetson AGX Xavier which is required to run the pose estimation script using the physical RealSense D435i, as well as the rosserial package as the interface between ROS and Arduino.

Challenges

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

1. Naming conflicts when including multiple cameras in one Zamboni's URDF: This is mainly due to links or joints that do not change their names based on the input arguments. This leads to conflicting namings when one includes two cameras' URDFs but both point to the same base file. This challenge was solved by replacing the fixed names of links and joints in the base file by variable names depending on input arguments (e.g. camera1_link, camera2_link). Also one important thing to note is that `args` is a global dictionary, which is accessible from within any file or macro, and this cost me hours debugging. For example, if one creates one single argument called `camera_name` in each of the macro that calls the base xacro file for the camera, and assigns it with `camera1` in one macro and `camera2` in the other, there will be errors such as "`camera1_link` is not unique" because `camera_name` is a global argument so both macros will pass the same text into the base xacro file.
2. Zigzagging waypoints generated using inconsistent leader pose estimations: Besides the causes mentioned above that led to zigzagging waypoints, the problem initially was much worse than what **Error! Reference source not found.** shows. It was because inside the pose estimation script, I iterated over each board, and for each board, I took two images from two cameras, determined which image to use based on the number of markers detected for that particular board, and broadcasted the transform from that board to the selected camera per iteration. This caused inaccurate pose being broadcasted even

when a board is in a bad configuration (such as the board on both sides when the leader is driving straight forward). Therefore, the solution was to maintain a maximum number of markers detected per board, besides the number of markers per camera in each iteration, and to skip any board that doesn't return more markers in either camera view. In this way, it's guaranteed that only one tf will be broadcasted per callback instead of three tf's in the worst case. This significantly reduced the amplitude of the oscillation in the waypoints generated to the extent shown in **Error! Reference source not found.**. And as mentioned, further smoothing is required for local waypoints.

3. Setting up environment on Jetson
   This challenge was mainly caused by the uncleaned environment from the team that used this Jetson before. For example, some packages that took huge amount of disk space will not be used by our project but removing them causes dependencies issues in other packages that we need. Also since Jetpack only supports up to Ubuntu 18.04, which means ROS Noetic cannot be installed, we have to use Melodic and Python 2, but a lot of packages and libraries have either stopped the support or had different installation approaches than the official ones, such as OpenCV and PCL. The solution was to clean the packages or dependencies that caused errors and to reinstall the ones we need through trial and error.
4. Unstable connection between Jetson and RealSense D435i
   This problem happens less frequently when testing the camera on our own laptop but it becomes much worse when connecting the camera with Jetson. There are errors showing the USB port cannot be found when launching the camera through ROS, while the GUI provided by RealSense always works like a charm, which is confusing. The current solution, which kills this problem 90% of the time, is to launch the camera along with the initial_reset argument and low FPS for both RGB and the depth streams, followed by unplugging and plugging the camera (N.B. it's important to re-plug the camera only after the launch file is launched).

Teamwork

- Nick continued to focus on the RC car and developed the PID speed control with the encoder installed. He also fixed the suspension and replaced the tires so that the chassis can support the stack without scraping the ground when moving. He also worked with Rathin to test the localization.
- Rathin secured the Husky robot from Prof. Kantor, found better wheels from other non-used RC platforms to replace ours, and took part in every aspect of

the RC car testing and validation. He also got the PCB working after I soldered the components.

- Jiayi kept refining the waypoint generation after integrating my perception. She actively tested it so that I was able to get feedback and changed the pose estimation script accordingly.
- Yilin integrated the localization and perception together on the RC car and solved any conflict on Jetson, which was challenging as well as exhausting. He also took an active part in testing the RC car, accomplished the very basic leader follower on the RC car with Husky as the leader, though it still required a lot of debugging.

Plans

Before SVD, I plan to refine the pose estimation algorithm whenever it causes any error in the real testing. I will also try to add tracking to reduce wobbling as suggested by Grace during the PR. Finally, I will be actively involved in the testing and validation of the functional and performance requirements.