Table of Contents

Individual Progress

My work in the past two weeks focus on the obstacle detection and visual odometry functionalities on Zamboni.

Obstacle Detection

I first continued from the LiDAR-camera calibration pipeline we left with two milestones ago. Due to the difficulty of setting up a proper, accurate calibration environment before using the self-automatic LiDAR-camera calibration package just as how we calibrated RealSense with VLP-16 on the ATV, we were not able to replicate that on Zamboni under the basement. Therefore, I resorted to the manual calibration tool instead. In particular, it requires an image and a point cloud file at the same timestamp. It shows up a GUI where one can manually tune the extrinsics (as well as intrinsics if not provided) until the projected point cloud onto the image plane matches the input image. This looked sufficient for our use case but due to the limited computers we have in our team, we were only able to launch and run the calibration tool on the MRSD computer. The problem coming with calibrating LiDAR and camera indoors was: the point cloud was very noisy in presence of multiple foreground objects, e.g. chairs, shelves, etc. When the point cloud was projected onto the image, one cannot easily identify point clouds that are associated with a particular foreground object. We had a hard time trying to match the point clouds with the image even before tuning the extrinsics. In light of the time constraints before FVD, we changed to a much simpler but yet effective enough approach to obstacle detection: running a detector on images from the RealSense. The disadvantage of this approach is that it cannot give the 3D position of the obstacle as we do not run the depth stream on RealSense. We will only be able to send braking commands whenever there's an obstacle detected in the current frame, regardless of distance of it from the ego. However, this is still an effective approach because according to our system requirements, we only need to detect obstacles within 10 meters and stop the vehicle 2 meters in front of the obstacle, which means as long as the camera resolution is good enough to detect obstacle within 10 meters and the vehicle is able to decelerate to a stop 2 meters in front of the obstacle, the system requirement can be validated.

I chose the MobileNet Single Shot Detector (SSD) because it is lightweight and does not require a GPU to run in real time on edge devices. To avoid the detector counting the leader vehicle (or its marker board attached) as an obstacle, we assume only two categories can be obstacles: person and chair. This is a valid assumption because during the intermission of an ice hockey game, persons are most likely the only

obstacle. The other possible obstacle is the gate, which is not in the training categories of the pretrained MobileNet, so we exclude it from our assumption but it's definitely a category we should consider adding as an extension of the project scope. Due to the wide FoV of RealSense, we need to limit the region of the image where detection happens in order to only consider obstacles that appear right in front of the vehicle. In other words, we don't want the Zamboni to stop if the obstacle appears outside the path it is following. To accomplish this, I simply cropped the image from its center and passed the 640x640 portion of the original 848x640 image into the detector.

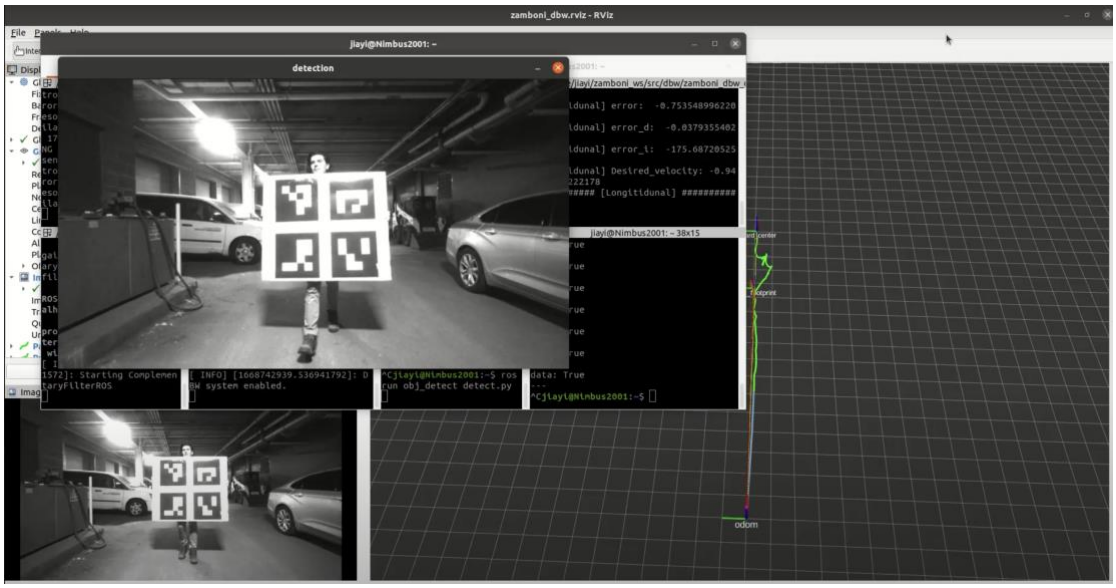

Figure 1. Object Detection does not detect person behind the board, which is as expected

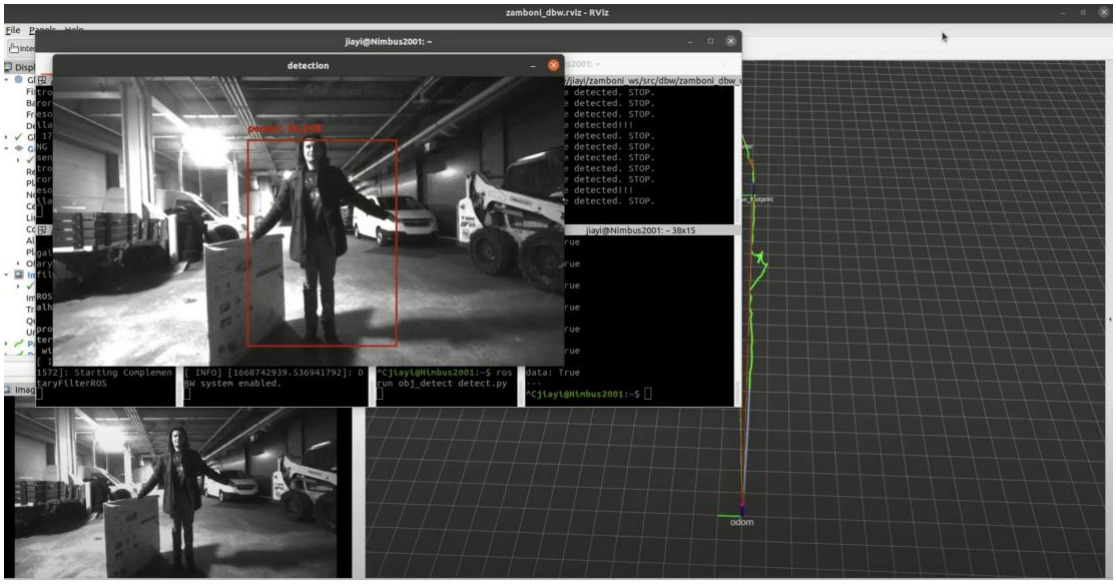

Figure 2. Object detection detects person with more than 90% certainty

Figure 1 and Figure 2 show the results of the obstacle detection.

Visual Odometry

When deploying our previous wheel odometry on Zamboni, we found two challenges: (1) the errors in the rotation of the IMU on RealSense D435i accumulate too fast and cause drift severely (2) reading RPM from the motors on Zamboni and converting to translation are not straightforward and prone to errors too. In light of these two challenges, we changed the wheel odometry to visual odometry that is independent from the hardware on Zamboni. We used the built-in Visual-Intertial Odometry (VIO) of RealSense T265i. It worked out of box because we already have the environment to run all RealSense cameras when we worked with RealSense D435i before. We published static transforms between (1) T265 odom frame and the Zamboni's odom frame (2) T265 pose frame and the Zamboni's base_footprint frame, so that the rest of the stack can keep untouched. The IMU on T265 is much more stable than D435i and even running a single Inertial Odometry gives good enough localization results in our use case. In summary, we currently run two cameras on the system, one is D435i that is responsible for obstacle detection and leader pose estimation in parallel, and the other one is T265 that is responsible for the vehicle's odometry only.

Challenges

The major challenges I have encountered are:

- One challenge we encountered when running the detector together with all other modules on Zamboni was that the frame rate was super low with significant delay. The problem did not occur when we solely test the obstacle detection itself. I analyzed it and found that even though the detector can run without a GPU, one forward pass still requires quite a lot computer resources. The delay was caused by the high frame rate of the camera (90 FPS) and the callback where detection happens runs whenever it receives an image. Therefore, if the callback cannot complete the forward pass within 0.011 seconds, which was what happened when we run the detection with other subsystems altogether, the incoming images will simply accumulate and causing increasing delay over time. I solved this by manually limiting the callback rate. In particular, I checked when was the last time the detector ran and I only run the detector again if it has passed 0.05 seconds. As such, I limited the callback rate to 20Hz, which is much lower than the frequency of the camera, giving a lot more time for the detection to finish per image. We ran this on Zamboni with other modules and the delay no longer happened.
- The other challenge we encountered when deploying the visual inertial odometry was the appearance of dynamic objects inside the image frame. To be specific, because T265 relies on both visual and inertial measurements to

get a robust odometry, it requires most parts of the image to include static (or rigid) objects/background only. If, say, a dynamic, non-rigid object is moving in front of it and taking up a lot of space in the image, the visual part of the odometry will fail even using RANSAC because there is not enough robust feature correspondences between two frames. Although this should not be a case in our project as the leader should be the only moving object all the time and it is more than 5 meters away from the ego, we still need to keep this in mind and when necessary, we should switch to Inertial Odometry only.

Teamwork

- Since Zamboni has arrived last week, all of us collaborated to test both lateral and longitudinal controllers on the Zamboni after integrating all subsystems together. The team also spent a lot of time testing the drive-by-wire interface. Rathin set up motor RPM for wheel odometry. Nick installed the mounts for LiDAR and camera. Rathin Nick, Yilin and Jiayi was involved in the final testing of the system on ATV to ensure our back-up FVD worked well.

Plans

I plan to help test the full system together with the team in preparation for FVD and FVD Encore.