Automated Driving Using External Perception

Individual Lab Report - ILR04
March 23, 2023

Team E - Outersense

Author:
Ronit Hire

Team Members:

Atharv Pulapaka
Dhanesh Pamnani
Jash Shah
Shreyas Jha

Contents

1 Individual Progress

To achieve the goals of this PR, I decided to work on designing and implementing the underlying ROS architecture for the perception pipeline. I also probed a bit into the controls part of the RC car and visualized the performance of PID at different levels of latency.

1.1 Perception

Given the dynamic nature of our system, it is important for us to separate the data processing aspect of various perception processes from the communication aspect. Using ROS as a middle-ware simplifies this but requires careful thought on how the callbacks and data-flow is structured.

I took to designing the data classes using OOP principles and a draft version of the UML diagram can be seen in the figure below.

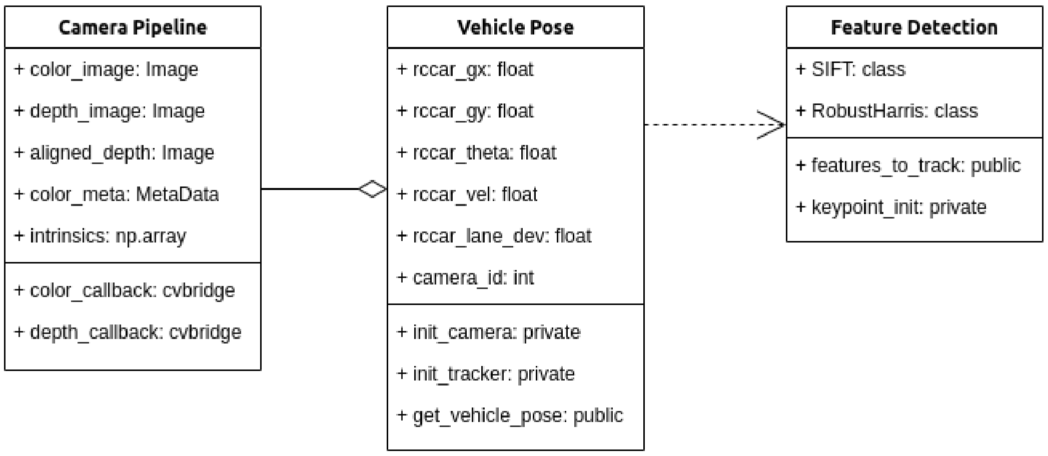

Figure 1: Class diagram for perception

The goal is to have a state variable for the RC car which is constantly updated by different perception blocks based on the new incoming data from the camera pipeline. For the camera pipeline, I decided to use the ros-wrapper provided by the realsense library because it does all the book-keeping activities like assigning time-stamps and the end user doesn't have to worry about those low level details. It also provides various launch files to enable post-processing functions on the raw-image. For state estimation, I decided to have global co-ordinates, orientation, velocity and lane deviation as the state variables. These will be updated by the object detection and tracking methods.

Having a well defined software architecture would help the team easily integrate different components together and also help us to make our codebase modular and extensible.

1.2 PID control

Building on top of the F1-tenth simulator from the last PR, I decided to replicated our track and vehicle at proper scales. The simulator uses gray-scale images coupled with a "yaml" config file to create a map of the environment. The track image can be seen in the figure below:

Figure 2: Grayscale image of track

I simulated the network communication latency between the perception and the control block by playing around with message buffers and loop rates in ROS. By increasing the incoming queue buffer size on the ROS subscriber and running it at a slower loop rate, I managed to delay the feedback for the PID controller.

The following conclusions can be drawn from these experiments:

- For 50-100 ms, the PID controller is able to keep the car within limits without much sway
- For 100-200ms, the car stays within the limits buts sways around a lot
- For 500+ ms, the car exceeds the lane boundaries

A snapshot of the experiments can be seen in the figure below:

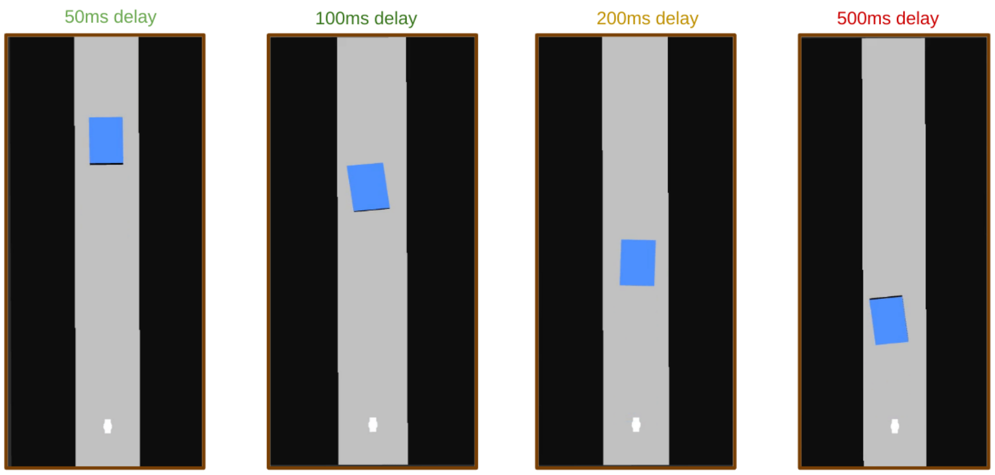

Figure 3: RC car iunder PID control

2 Challenges

2.1 Perception Pipeline

One of the major setbacks in setting up the perception pipeline was realizing that ROS does not work well with a conda environment. ROS relies on packages provided by the original distribution and any python libraries installed in a conda environment are not accessible to ROS. I needed a specific version of openCV for the object recognition and tracking methods but ROS-noetic comes with an older version which does not have those. I wanted to use a locally compiled version of opencv and it proved very difficult to get it running. I had to modify various environment variables and recompile opencv with python bindings to make it work. Also, ROS nodes which are initialized on the realsense camera follow a specific convention for topic names and we wish to modify those names to reflect a unique id for each of our infrastructure unit. The suggested methods to achieve that involve changing parameters in the launch files but that is not working out and it is something we have to figure out going ahead.

2.2 PID control

The way in which tracks are defined in the F1-tenth simulator allows for rapidly testing different configurations but representing actual track as a gray-scale image can sometimes prove to be tedious. The real-world dimensions are based on pixel distance and we need to ensure that the mapping remains uniform throughout the image. Also, the simulator defines an occupancy grid by thresholding the intensities in the image (the darker pixels are marked as obstacles and lighter ones act as free regions). Accurately defining our track as an image with the correct intensity level was time consuming and it took me multiple iterations to get it right.

3 Team Work

The work for perception, controls and car customization is allocated to specific team members with other members supporting them in an ad hoc manner. Most of the heavy manufacturing was completed during the course of the previous PRs and for this PR only light finishing tasks were carried out.

- **Jash Shah**: Jash completed marker detection on the RC car. The output of his activity can now feed into the tracking code for velocity estimation. He also worked on a calibration script for the RGB Intel Realsense camera using a checkerboard and the ROS calibration tool.
- **Shreyas Jha**: Shreyas worked on multi-device ROS setup for the car and is able to achieve robust tele-operation of the car via laptop. With that completed, the RC car is now in a state to accept commands from a controls block. He has also started work on time synchronization with NTP.
- **Dhanesh Pamnani**: Dhanesh completed the assembly of all the three infrastructure units. He helped Jash with the calibration tasks and is now working on a validation procedure to ensure that the all the extrinsics and the intrinsics obtained via calibration are reliable and usable in other blocks of perception.
- **Atharv Pulapaka**: Atharv worked on testing his MPC implementation for different starting configurations of the RC car and also tried to incorporated sensor biases and noise into his python simulations. He is working on on-bench steering tests for the RC car and will soon move onto on-track testing.

4 Plans

Moving ahead, my individual goals are:

- Implementing velocity estimation via object tracking.
- Literature review of robust velocity estimation methods from camera data.
- Structuring all the perception blocks into a modular and configurable ROS pipeline.
- Improving the implementation of the PID controller by adding feed-forward terms.
- Setting up Jetson Nano to act as a ROS node for pose estimation.

As a team our goals are:

- Corroborate results of camera calibrations with other methods.
- Estimate the pose of moving RC car from infrastructure mounted camera.
- Incorporating data from multiple infrastructure units.
- Communicate velocity and steering profiles to ROS nodes on the RC car.