Automated Driving Using External Perception

Individual Lab Report

MRSD Project Course I

Spring 2023

Shreyas Jha

Team E: OuterSense

Teammates: Atharv Pulapaka, Dhanesh Pamnani, Jash Shah, Ronit Hire

ILR03

Mar. 1, 2023

Contents

Individual Progress

Previously, I enabled teleoperation of the RC car using a ESP32 microcontroller board and the Blynk IoT platform. Over the last two weeks, my goal was to work on enabling teleoperation via ROS messages so that the RC car can easily be integrated with the other subsystems. I finalized the on-board electronic hardware to include a BLDC (Brushless DC) motor, 6-cell LiPo battery, servo motor, 30A ESC (Electronic Speed Controller), Raspberry Pi 4B, a voltage sensor, an IMU, and a PDB (power distribution board). The major components of the RC car are shown in Figure 1.

I worked with Atharv to develop the conceptual design for the PDB. We shifted to the Raspberry Pi so that we can easily develop the software on the RC car as ROS nodes. We chose work with the ROS Noetic distribution since it has more support for the Intel RealSense camera (our primary perception sensor). To run ROS Noetic on the Raspberry Pi, it was recommended to use a Linux 20.04 distribution instead of the Raspbian OS. I installed an Ubuntu 20.04 server and ROS Noetic on the Raspberry Pi.

I then installed the required GPIO libraries and worked on a ROS node to read and execute the desired motor RPM and steering angle. I used a hardware PWM on the Raspberry Pi to control the speed of the BLDC motor by sending the signals to the ESC. From our previous execution of teleoperation, we found that we would like more fine-grained control over the motor RPM. Thus, instead of using an existing servo library, I worked on calibrating and controlling the ESC from scratch. I then developed the teleoperation node, which serves as the UI to accept commands via the keyboard to set the speed and steering angle of the RC car. I also attempted to deploy a multi-machine ROS architecture, with the teleoperation UI node deployed on a laptop and the driver node deployed on the RC car.

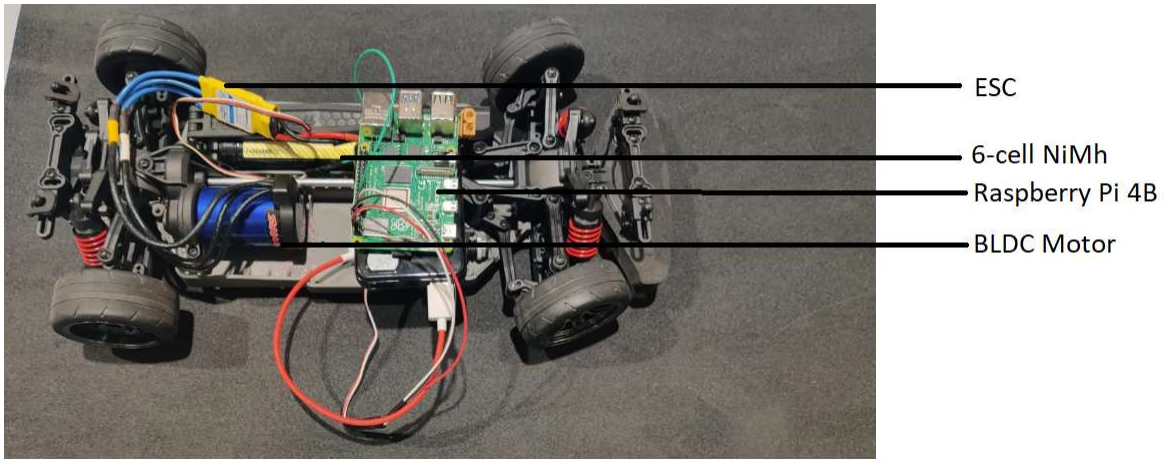

Figure 1: RC car subsystem with Raspberry Pi 4B

Challenges

Installing a desirable OS for Raspberry Pi: The preferred OS for the Raspberry Pi was UbuntuMate 20.04 for a 32-bit ARM architecture. UbuntuMate is a light-weight OS and offers support for GPIO control on the Raspberry Pi. I faced challenges whilst trying to install this, and its variants on the Raspberry Pi. However, I could not find a fit for our Raspberry Pi and UbuntuMate 20.04, as they were either flagged outdated or not detected by the Raspberry Pi. I decided to instead work with an Ubuntu 20.04 server and then connect to the Raspberry Pi via SSH to develop the required software.

Running ROS over multiple machines: I deployed the driver node on the Raspberry Pi and the teleoperation node on my laptop, connected both devices onto the same WiFi network, defined the ROS master, and launched the nodes. I could detect that the nodes were running on both the machines, and a topic was supposed to be connecting them. However, the communication over the topic was not occurring. The issue was narrowed down to be with the communication, since when both the ROS nodes were deployed on the Raspberry Pi, the desired functionality was achieved. I followed the instructions on the MRSD Wiki and the official ROS Documentation to try and resolve this.

Teamwork

Atharv Pulapaka: Atharv has been primarily working on implementing the control subsystem using model predictive control. He moved from Simulink into a Python environment, and was studying the impact of communication latency in the system by modelling it in different ways. I collaborated with him to work on the PDB for the RC car.

Dhanesh Pamnani: Dhanesh has been owning the fabrication, machining, and assembly of the hardware for the track and the infrastructure units. He assembled two additional infrastructure units. He also worked with Jash to 3D-print a mount for the Intel RealSense camera and performed stereo calibration using the Intel RealSense software.

Jash Shah: Jash has been working on refining the pose estimation method. He developed an algorithm to detect a square-shaped marker (which we can paste on the hood of the RC car) and estimate its yaw angle from a bird's-eye view. He also worked on the design of a mount for the camera and RGB camera calibration.

Ronit Hire: Ronit has been studying and experimenting with a simulation made available by the F1/10th autonomous racing community. This is developed using ROS and we see value in customizing this to our needs to test different approaches to our control architecture. Whilst Atharv has been experimenting with MPC, Ronit has been developing the control subsystem using PID control.

Plans

I intend to debug the issue with the multi-machine ROS architecture and complete the RC car subsystem so that it is robust and ready for integration. I intend to work on the following:

1. Resolve the issue with the multi-machine ROS architecture.
2. Optimize the communication framework by diving deep into the WiFi and ROS topic configurations.
3. Study and implement time-synchronization over multiple machines via ROS.
4. Finalize and procure electronic hardware for the infrastructure sensing units.