



# **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

ILR04

Mar. 22, 2023



**Carnegie  
Mellon  
University**

Contents

Individual Progress..... 1

Challenges ..... 1

Teamwork ..... 2

    Atharv Pulapaka ..... 2

    Dhanesh Pamnani ..... 2

    Jash Shah..... 2

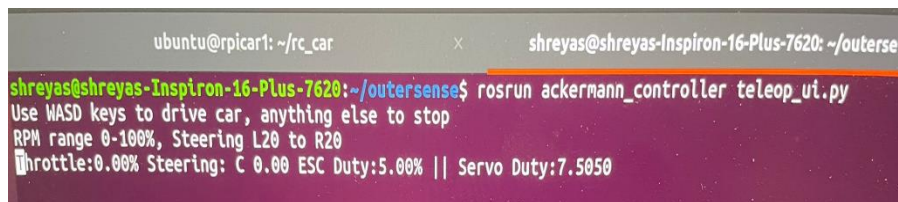
    Ronit Hire ..... 2

Plans..... 2

## Individual Progress

Previously, I developed the software for the RC car as ROS nodes and deployed them on the Raspberry Pi. I continued from there, working on setting up the disturbed network architecture to enable fast and reliable communication between various subsystems. I set up a dedicated 5GHz WLAN network using the TP Archer Link high-speed router and configured the DHCP settings to allot static IP addresses to the devices in the OuterSense system. I then configured the ROS Master on my laptop and configured clients accordingly. Post this, I deployed the teleoperation UI software on my laptop and the hardware controller on the RC car. By using the 'W' and 'S' keys a user can incrementally increase and decrease the speed of the RC car respectively, which maps to the configured duty cycle range for the ESC. Similarly, by using the 'A' and 'D' keys a user can incrementally steer the RC car left and right maps to a duty cycle range for the servo motor actuating the Ackermann steering mechanism. The 'C' key allows a user to center the steering instantaneously, without changing the speed. All other keys were configured to act as E-stop buttons, halting the car instantaneously. This [document](#) contains all the steps required to connect a different device to the network, configure a static IPv4 address, and teleoperate the RC car. Figure 1 shows the user interface.

I then worked on calibrating the UM7 IMU, by connecting it to a converter so the IMU can directly be connected to the laptop for calibration via the OEM's software. I was able to successfully calibrate the IMU, however there still is a slight drift in the yaw values. Finally, I began studying time synchronization methods and experimented with NTP.



```
ubuntu@rpicar1: ~/rc_car x shreyas@shreyas-Inspiron-16-Plus-7620: ~/outersense
shreyas@shreyas-Inspiron-16-Plus-7620:~/outersense$ rosrund ackermann_controller teleop_ui.py
Use WASD keys to drive car, anything else to stop
RPM range 0-100%, Steering L20 to R20
Throttle:0.00% Steering: C 0.00 ESC Duty:5.00% || Servo Duty:7.5050
```

Figure 1: UI to teleoperate the RC car over ROS using custom WLAN network.

## Challenges

**Jitter in servo motor:** Even with a constant duty cycle and hardware PWM duty cycle to the servo motor we observed a random jitter in the steering. This is a risk for the system. We need to conduct more on-ground tests to evaluate the impact of this. If it is found to be significant, we may have to first try a different servo motor or move to a more reliable actuator like the DYNAMIXEL motors.

**Time synchronization between disturbed systems:** This is a real challenge in robotic systems and is critical to accurate sensor fusion and tracking for control inputs in our system which has inherent latency due to communication of the data packets over the network. Our systems do not have RTCs (Real-time clocks), hence PTP (Precision Time Protocol) was not an option for now. On trying to configure NTP (Network Time Protocol) such that client devices track the time on the ROS Master I faced an error which said that the connection to the server failed. I followed the instructions on the MRSD Wiki and the

official ROS Documentation to try and resolve this. However, this requires a further deep dive and debugging now to synchronize our system clocks, at least in the order of milliseconds (possible over NTP).

## Teamwork

**Atharv Pulapaka:** Atharv worked on updating the control subsystem to output a sequence of control inputs which is critical to our system architecture. He also appended a profile to ensure that the car comes to a smooth stop if a new command is not received within a threshold (loss of communication between the central decision-making system and the RC car) to prevent runaway robots. I collaborated with Atharv to improvise the hardware controller and perform HIL (hardware in the loop) testing. He also worked on developing the PDB for the RC car and identifying vendors for the required components.

**Dhanesh Pamnani:** Dhanesh has been improving the calibration process of the Intel RealSense perception sensors. He also worked with Ronit to understand the perception pipeline and refactor the code.

**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. Additionally, he and Dhanesh also began experimenting with the lane detection algorithm.

**Ronit Hire:** Ronit completed the simulation tests of the PID architecture by modeling our specific subsystem parameters and verifying the performance. He then defined the perception pipeline to refactor the existing developments into a scalable implementation. He has also been working on a method to estimate the velocity of the RC car using the overhead cameras.

## Plans

I intend to work on the following:

1. Develop a feedforward controller for the RC car to track the acceleration profile based on the IMU acceleration estimates.
2. Collaborate with Jash to set up the embedded hardware for the infrastructure sensing units.
3. Attempt to electronically reverse motor rotation using 2 channel relay.
4. Implement time-synchronization using NTP over multiple machines.