OuterSense

Automated Driving Using External Perception

Team: Ronit Hire, Dhanesh Pamnani, Shreyas Jha, Jash Shah, Atharv Pulapaka

ILR10

15th November, 2023

Sponsor: Nissan Automotive Inc.

Liam Pederson, Najam Baig, & Viju James

THE ROBOTICS INSTITUTE

Carnegie Mellon University

Table of Contents

1. Individual Progress

At this phase of our project, the primary focus shifts towards comprehensive integrated testing and fine-tuning of system parameters, with reduced emphasis on design and architecture development. Our efforts are directed at executing structured test plans aligned with the upcoming fall validation demo. Initially, we conduct single-car testing, integrating all systems and running the vehicle in a loop for 6 to 7 laps. Continuous monitoring of perception, state estimation, planner, and control performance is performed throughout this phase. Subsequently, we extend testing to include another car, ensuring that both vehicles meet performance requirements before progressing. Scaling up, we advance to testing with two cars in a loop, running the system for an extended duration of 15-30 minutes. The evaluation involves scenarios with vehicles moving at both identical and different speeds to assess system performance, especially at intersections and during cruise control. Any deviations from desired performance trigger parameter tuning to optimize system behavior.

1.1 Asynchronous callbacks

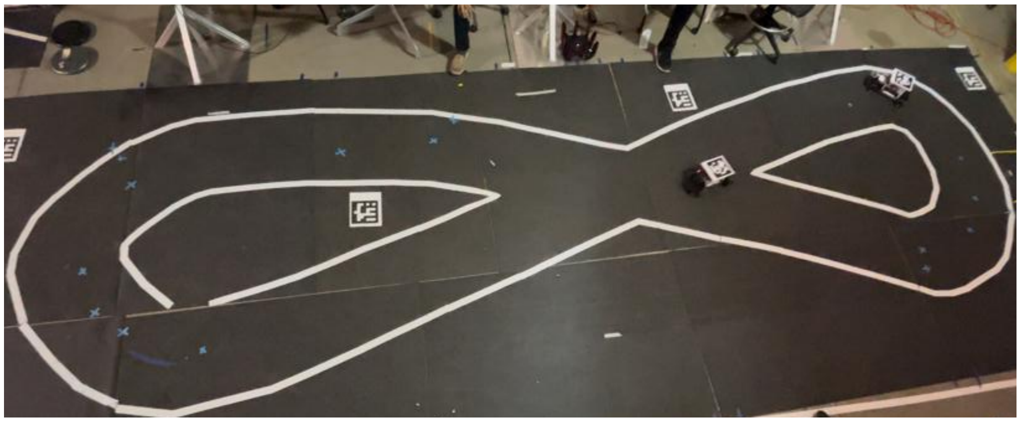

Figure 1 : Integrated test with 2 cars

In the control system architecture of our ROS-based system, asynchronous operations are crucial for maintaining responsiveness and efficiency. The control system integrates inputs from both the planning system and state estimation system, each facilitated through their respective ROS callbacks. Notably, the planner callback checks for cruise control and intersection modes. In addition to these event-driven callbacks, the control system features a timed callback responsible for executing Model Predictive Control (MPC) and publishing control commands to the RC vehicle.

An observed anomaly arose during cruise control scenarios after multiple loop runs. We noted instances where the trailing vehicle would occasionally receive full-speed commands from the MPC for a one timestep (small fraction of a second) , resulting in noticeable jerky motions when the vehicle was intended to come to a complete stop behind another vehicle. Extensive debugging and input monitoring revealed that the issue stemmed from a timing misalignment between the planner callback, responsible for updating the vehicle's speed, and the timed callback executing MPC. Specifically, the MPC occasionally processed with a default speed set in the code before the planner callback could update it to 0 for stopping.

To address this synchronization challenge, we implemented a threading lock using threading.Lock() to ensure that other tasks could only proceed when the timed callback for MPC execution or planner callback had completed. This solution effectively mitigated the timing misalignment issue, ensuring consistent and reliable behavior during cruise control scenarios. The lock was acquired at the beginning of the callback and released upon completion, preventing concurrent access to critical sections of code and ensuring proper sequencing of operations.

The **threading.Lock()** in Python is a synchronization mechanism provided by the threading module. It is used to control access to a shared resource or a critical section of code, ensuring that only one thread can access it at a time. The basic idea is to prevent multiple threads from simultaneously executing a section of code that might lead to data corruption or undesired behavior.

Figure 1.1 shows track for integrated testing.

1.2 Restructure cruise and intersection checks

An unintended case surfaced when the planner unexpectedly ceased operation due to an unrelated issue, revealing a critical scenario. In this specific situation, a vehicle found itself within an intersection, and the planner discontinued sending commands. The control system, designed to suspend command publication in the absence of pose and plan inputs for safety reasons, adhered to this behavior. However, the challenge emerged as the intersection checks were exclusively confined to the planner callback. With the planner's functionality interrupted, the vehicle failed to communicate its presence in the intersection to other vehicles. Consequently, an approaching vehicle, lacking this pertinent information, proceeded without halting, resulting in a collision with the stationary vehicle.

To rectify this situation, we concluded that the responsibility for these safety checks should be shifted from the planner callback to the control callback, responsible for executing Model Predictive Control (MPC) and publishing commands. The control callback, operating continuously as a timed callback, ensures consistent execution of critical intersection checks before generating steering and acceleration commands from the MPC, as well as publishing them. This adjustment ensures that the system maintains situational awareness, even during temporary halts in specific callbacks, preventing collisions and reinforcing safety measures within the intersection environment. For further details on the intersection logic, please refer to our previous report.

2. Challenges

2.1 MRSD project

- Currently challenges are mainly with the planner, it is a difficult task to very accurately map the real world in the planner system. It is important because we need to ensure the vehicle remains in the center of the lane and does not leave lane boundaries. We are in the process of tuning the scaling parameters to represent the real world as closely in the map referred by the planner.

- Another challenge is with waypoints, we first have a manual run where we drive the vehicle manually and record pose of the vehicle. This pose data is used to create waypoints for the planner, any fault detection with pose measurements of this manual run will cause the planner to perform undesirably with wrong waypoints. We are finding alternate ways to get waypoints and also conducting some sanctity checks on the waypoints to discard wrong ones.

3. Team work

Ronit Hire: Ronit worked on ensuring perception units are able to detect and track multiple cars. He generated unique IDs for each vehicle that helped the control system to associate data with respective cars. He is currently working on ensuring the perception system is more robust. We both conduct integrated testing and check for failure cases.

Shreyas Jha: He has worked on fusing camera data with odometry and IMU to get more accurate state estimates. He also worked in resolving the low level RC control issues to ensure the car followed the command given by MPC. We worked on fine tuning RC cars to follow speed and steering commands correctly. We also conducted integrated tests to ensure system robustness.

Dhanesh Pamnani: Dhanesh and Jash developed the planner system that gives collision free paths for controllers to follow. We tested the planner with other systems and identified places where planner logic needed to be changed. He is currently working on solving existing problems regarding the planner and then will test again with other systems.

Jash Shah: Jash is working on the planning subsystem, he and Dhanesh both are involved in planner development. We all are conducting integrated testing and ensuring planners behave desirably. He has worked on the planner to handle static obstacles and generate alternate paths to avoid collision.

4. Future Plan

- The next step is to add dynamic obstacles and check system performance.
- Continue with existing testing with all systems with static obstacles and only controlled cars to improve performance