Automated Driving Using External Perception

Individual Lab Report - ILR05
April 6, 2023

Team E - Outersense

Author:
Jash Shah

Team Members:

Atharv Pulapaka
Dhanesh Pamnani
Jash Shah
Ronit Hire
Shreyas Jha

Contents

1 Individual Progress

1.1 Perception

As for the MRSD project I have mainly been working on the perception front. Under perception, I worked on two facets; marker detection as well as running the perception pipeline on embedded devices.

1.1.1 Marker Detection

The previous method for marker detection used was detecting a simple HSV-based color detection algorithm with area and shape filtering. However, there were a few problems with this method,

1. **Variance to light and shadow**: The HSV method used for filtering color used to fail when a small shadow or change in lighting occurred. This was because the main parameter to be tuned was ”H” or the Hue. Giving a very large range for this parameter is used to lead to false positive detections whereas giving a very small range would not be plausible because a change in any illumination would cause the detection algorithm to fail.

2. **Slow detection**: The perception pipeline runs on 2 different threads; tracking and detection. The detection thread runs at a slower rate than the tracking pipeline. This is done because the tracker drifts after a few frames and the detection algorithm brings it back or re-initializes it so that the tracking is accurate again. HSV-based filtering methods are extremely slow and computationally inefficient to be run even at a slower rate. Hence, we had to move away from it.

3. **Fewer key points being detected by the tracking pipeline**: By using the HSV detection method, the number of key points being detected was reduced significantly, which in turn made the velocity estimation inaccurate. Hence, we decided to pivot from this method.

As can be seen in Fig. 1, the current marker being used is a square identifier that suffers from the aforementioned problems.

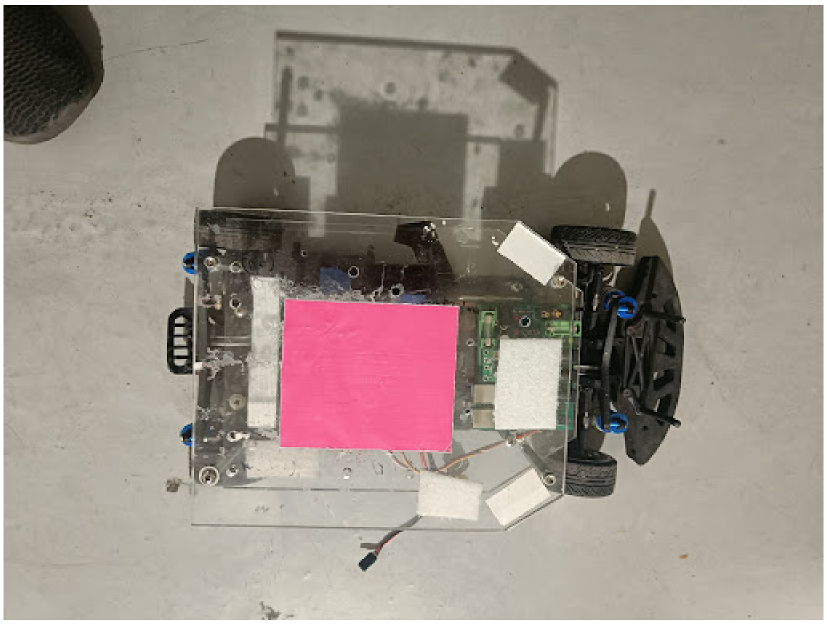


Figure 1: Current method (HSV-detection)

We decided to pivot to a more robust and fast rotation-invariant template matching algorithm to detect the marker from a birds-eye view. The advantages of using this method were as follows:

1. **Faster detection**: This method enabled much faster tracking than HSV-based detection
2. **Robust to changes in lighting conditions**: Because the method being used was simple template matching, there was no change due to lighting conditions, the method works on simple gray-scale images.

This method, however, still encounters the problem of fewer key points being detected. A better marker with more features can be used to overcome this problem. This occurs because our track, as well as most of the elements of the car, are black. Hence, the camera is not able to differentiate between the base of the car and the track. As shown below in Fig. 2, the current method being used is a circular marker in a gray-scale image.

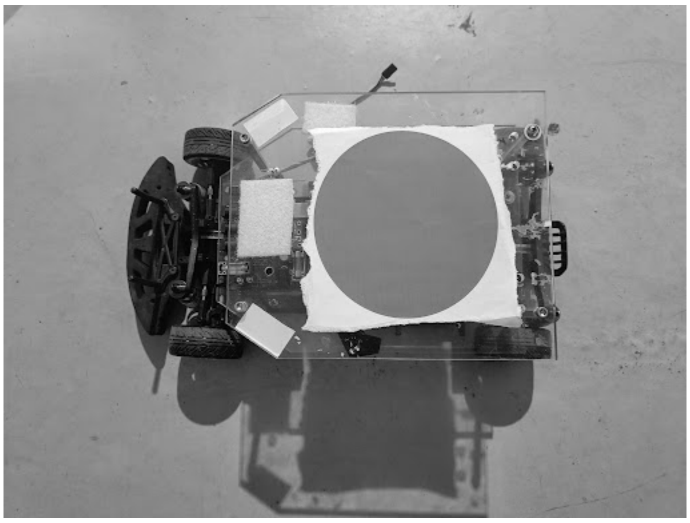
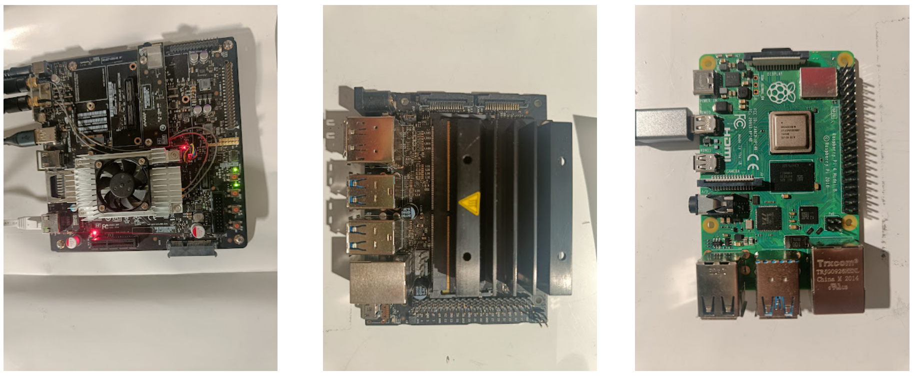


Figure 2: New method (Template-matching)

1.1.2 Embedded Systems Testing

The initial plan was to use edge computing on the infrastructure units in order to streamline data that was being sent to the master computer. To do so, we required some form of embedded system. We tested a range of embedded systems as can be seen in Figure 3. However, there were a number of problems with each of the embedded systems. More information about the challenges is in Section 2.1.2. However, the conclusion of the testing was that for the Spring Validation Demo, we will be going ahead with computers as the edge devices, not just because of the computing power but also because the architecture supports all the libraries and modules we wish to use.



(a) Jetson TX2 **(b) Jetson Nano** **(c) Raspberry Pi**

Figure 3: Range of embedded systems tested

2 Challenges

2.1 Perception

There were 2 main issues that came up in the perception subsystem.

2.1.1 Marker Detection

As mentioned in Section 1.1.1, there were several problems with the previous marker detection algorithm being used. However, I was able to solve most of them using a circular marker. Further, another interesting problem we faced with using template-matching was that there were very few fast algorithms that were rotation-invariant. To counter this, I used a circular marker so that it can be detected from any angle. This workaround was used so that the marker can be used at any yaw.

2.1.2 Embedded Systems Testing

The problems with embedded systems were as follows:

1. **Architecture mismatch:** The code that was written for the perception pipeline was written on a laptop that has a 64-bit architecture. However, the Jetsons have an ARM architecture which work on with a different protocol. The Raspberry Pi also has an ARM architecture, but because the CPU of the RPi was a little more powerful, our code did work to an extent on it. However, when the load increased even a little, the RPi also used to heat up too much and drop frames.

2. **Different Ubuntu and ROS Version:** Officially, the Jetson Nano does not support any version of Ubuntu after 18.04. Similarly, it is tricky to flash a stable version of Ubuntu 20 on a Raspberry Pi. This means that the only version of ROS that can work on these systems is ROS Melodic. However, we used ROS Noetic to develop our code. Using ROS on different versions is unnecessary and hence we avoided it altogether. Even though I was able to get Ubuntu 20 working on both of these embedded devices, it led to problem number 1 (Architecture mismatch).

3. **Different version of OpoenCV required:** The version of OpenCV and CVBridge was also an issue and had to be built very delicately from source so that each faction of our perception pipeline worked well. Even though this worked for a short period, it would fail when more complicated libraries would have to be downloaded and run on different kinds of architectures. Hence, we decided to move away from this.

3 Teamwork

As for the teamwork, we are currently focusing mainly on the integration of the entire code and making the system run on a central node by taking feedback from perception.

- **Ronit Hire**: Ronit mainly worked on the perception pipeline. He wrote the base code for the entire structure of the perception unit. He also actively worked on improving the velocity estimation of the vehicle from a birds-eye view. Moreover, Ronit collaborated with me to test the edge-compute devices. Furthermore, he also handled a large faction of the logistics of the team.

- **Shreyas Jha**: Shreyas worked on making the VESC function with the RC car. He used ROS to integrate commands which he would receive from the MPC block. Moreover, he also studied documentation of VESC to make sure that the hardware is in order to stably operate the vehicle under all conditions.

- **Dhanesh Pamnani**: Dhanesh worked primarily on the RC car. He built the mechanical modules for the RC car which would house the Raspberry Pi, the VESC, and the battery. Moreover, he also attempted to reduce the error we are currently getting from the Intel Realsense cameras. Moreover, he developed the lane detection algorithm which is being used to provide waypoints to the MPC.

- **Atharv Pulapaka**: Atharv worked on the MPC unit. He mainly worked on integration with the perception pipeline to use data that can be used to create a closed-loop feedback system. He also modified the MPC block to take inputs as they were being received on the ROS topics.

4 Future work

4.1 Personal

On a personal front, I will be working on integration. Currently, there are a few problems with the velocity block of the perception unit. I will work on making this module more stable.

1. Make the template matching algorithm more stable.
2. Develop a simple averaging filter that will make the velocity estimation of the vehicle more stable.

4.2 Team

As for the future work of the team, we wish to achieve the following goals.

1. Integrate the perception pipeline with the controls block
2. Conduct several tests to find edge cases
3. Create filters that make the car travel much smoother than the current state
4. Work on multi-camera-based pose estimation