VROOM / auth

Virtual Route Optimization Mobility
Aristotle University of Thessaloniki

Bosch Future Mobility Challenge

ARISTOTLE UNIVERSITY OF THESSALONIKI

# Technical Report #2


**Authors**

Theodoros Papafotiou [*Team Leader*]

Efthymia Amarantidou [*Team Member*]

George Koutroumpis [*Team Member*]

Nikolaos Tsachouridis [*Team Member*]

**Responsible Mentor**

Emmanouil Tsardoulias
Antonios Dimitriou


**Date**

February 12, 2021

# 1 Introduction

The current report summarises the technical progress of the team V.RO.O.M. (Virtual Route Optimization Mobility) in the Bosch Future Mobility Challenge 2021 until January, 24th, 2021. Particularly, the activities planned during the time-period between the 21th of December 2020 and the 24th of January 2021 and their current status, as well as the upcoming activities are described shortly.

# 2 Planned activities

The planned activities during the aforementioned period, additionally to the team members in charge for each activity are shown below:

- Perception
  - **Traffic Signs & Lights Detection**
    *@Theodoros Papafotiou & Efthymia Amarantidou*
  - **Pedestrian Detection**
    *@Efthymia Amarantidou & @George Koutroumpis*
  - **Vehicle Detection**
    *@Efthymia Amarantidou & @Theodoros Papafotiou*
  - **Intersection Detection**
    *@Nikolaos Tsachouridis*
- Control
  - **Lane keeping**
    *@George Koutroumpis & Nikolaos Tsachouridis*
- Working tools
  - **Final track setup using the provided 3D printed parts of the original track and traffic signs**
    *@Efthymia Amarantidou & Theodoros Papafotiou*
  - **Final code configuration for RPi4**
    *@Theodoros Papafotiou*
  - **Simulation setup and error fixes**
    *@All members*

# 3 Status of Planned Activities

## 3.1 Lane Keeping [Done]

### 3.1.1 Goals

The goal of our algorithm is for the car to be bound between the lines of a lane.

### 3.1.2 Implementation

For our Lane Keeping algorithm we employed the Python API OpenCV.

- Processing of lines to determine left and right lane lines.
- Creation of average lines based on the categorized line segments.
- For the lane keeping part of our algorithm, there are two possible inputs from a frame: one line or two lines are detected.
  1. One line: the slope (therefore) angle of the line is used, slightly modified to not have abrupt steering angles.
  2. Two lines: an average steering angle is calculated based on their slopes and positions.
- The calculated angle is stabilized, until the PID controller is implemented.

### 3.1.3 Issues

The main issue encountered is when the vehicle is moving at high speeds and only one lane line is detected. Specifically, there is a danger of going off stage, due to very sharp and quick turns, or there it may not have enough time to make the full turn. There are a couple solutions, like getting a percentage of the full angle of the lane, or when a sharp turn is detected to lower the car's speed. Both work, and the one that will be in the final code will be decided when tests on the actual car and stage are made.

## 3.2 Traffic Signs and Lights Detection[Done]

### 3.2.1 Final approach

The final version of the traffic signs and lights detection script has been implemented using the Tiny-YOLO algorithm.

### 3.2.2 Implementation

- First version of the dataset was grouped in 5 different datasets, containing similar traffic signs.
- Models were trained on Google Colab using the Tiny-YOLO ("You Only Look Once") algorithm.

### 3.2.3 Issues

On our first approach, YOLO algorithm with a **single** dataset were used to train the model, which needed a lot of time to complete. Furthermore, testing indicated that YOLO algorithm is not fast enough to run on embedded devices such as the Raspberry Pi. To get over these problems, we decided to group our data, create 5 **separate datasets** and train our new models based on **Tiny-YOLO** architecture, which has a small model size and is suited for embedded computer vision devices.

## 3.3 Pedestrian Detection[Done]

### 3.3.1 Goals

The goal of our algorithm is to detect pedestrians either in between the road limits or in a randomly on track.

### 3.3.2 Implementation

For our Pedestrian Detection algorithm we took advantage of the openCV built-in method to detect pedestrians, which uses a pre-trained **HOG** (Histogram of Oriented Gradients) & **Linear SVM** model.

- HOG person detector is initialized.
- HOG Descriptor's *detectMultiScale()* function is used with the appropriate parameters to detect pedestrians.

- Detection area is determined, in order to know whether we need to react to this pedestrian or not.

## 3.4 Working Tools [Done]

**Final Track Setup** Parts of the final track (intersection, roundabout, straight line, curved line, parking-spot) were created using model-paper, blackboard sticker for the details and white duct-tape for the lanes. In addition, all traffic signs and lights were 3D printed.

**Code Configuration in RPi4** Based on the code of the BFMC startup_project, we made adjustments on the code and added the necessary files, in order to configure the workspace for implementing our Perception, Action planning, Behaviours and Control, according to our Software Architecture.

## 3.5 Activities under [Development]

| Activity | Goals | Implementation |
|---|---|---|
| Vehicle Detection | Recognizing static and dynamic vehicles, in order to incorporate them in the decision process. | Cascade Classifier |
| Intersection Detection | Recognizing intersections on the track and measuring the distance from the vehicle to the line. | ◇ The area of interest is masked and all possible horizontal lines are extracted from every frame<br>◇ An algorithm that merges the detected lines into one has been developed<br>◇ An algorithm to correlate the dimensions of this line (in pixels) to the distance (in mm) between the vehicle and the line has been implemented |

# 4 General status of the project

In general, a partially final approach of the car's perception was implemented and all scripts unit and functional tests were executed. After solving all issues with the Gazebo simulation, the team verified the effectiveness of each implementation in the actual simulation. In addition, the final software configuration for the perception and control on RPi4 is completed and the scripts are currently being integrated also on the car, which is placed on the created parts of the final track in laboratory facilities.

# 5 Upcoming activities

According to the suggested timeline of the team, during the period between the 20th of December 2020 and the 24th of January 2021, the upcoming activities are:

- Perception Integration
  - Roundabout detection
  - Ramp detection
  - Route planning
  - Intersection detection [Simulation and Finalization]
- Control
  - Intersection Navigation [Simulation and Finalization]
  - Roundabout Navigation [Simulation and Finalization]
  - Reaction to static elements
- System Integration
  - Total code integration from simulation to real-life car
  - GPS tracker integration
  - Traffic Lights interaction