Autonomous Vehicle Driving 2020/2021 - Group n°21

Ardit Nogu
Department of Information Eng.
University of Salerno
Italy
a.nogu@studenti.unisa.it
0622701118

Joel Haupt
Department of Information Eng.
University of Salerno
Italy
j.haupt@studenti.unisa.it
ERASMSIN02419

Antonio De Luca
Department of Computer Science
University of Salerno
Italy
a.deluca72@studenti.unisa.it
0522500885

1 SUMMARY DESCRIPTION OF THE PROJECT

For the project we were provided with the CARLA-simulator, a baseline of an autonomous vehicle driving algorithm and a traffic light module. The project consisted of two main tasks, the integration of a traffic light detection and an obstacle detection. In addition some minor adjustments in the main file and the behaviour planner had to be made to integrate the newly added functions. The first step was to familiarize oneself with the modules of the baseline and the traffic light detection in order to understand the mechanisms behind the implementation. The codebase uses a hierarchical structure with a mission planner, that plans the high level path on the map, the behavioural planner which is applied by means of a finite state machine that indicates the current state for safe behaviour, this state is passed on to the local planner which account for the velocity profile generation and the path planner and finally the controller that accounts for the movement and steering (longitudinal and lateral movement) of the car.

To implement the obstacle detection we had to program a new function that we split in three subfunctions. The first function calculates the next position of a moving object based on the current position, current velocity, its orientation (yaw-angle) and the temporal difference between each iteration step. As shown in Fig. 1 below the bounding box of the car is referred to as 3 circles that imply, if there is an obstacle inside of any of them we have an underlying collision. Thus the next function accounts for the three center points of the circles of the vehicle. In order to get the correct circle positions we had as a reference point the current position of the vehicle and its orientation which we used to get the circle in the front and in the back. For the pedestrians, one circle around them was sufficient as a bounding box. With the received center points of the circles we were finally able to calculate the distances between the ego-vehicle to any moving obstacle and thus obtain any possible collision. We then extended the function to detect collisions that can happen in the future by calculating a few iterations ahead. The obstacle detection relies on a rather limited formula that only gives an estimate of the future vehicle positions, therefore it is never exact. However, as a detection measure it suffices.

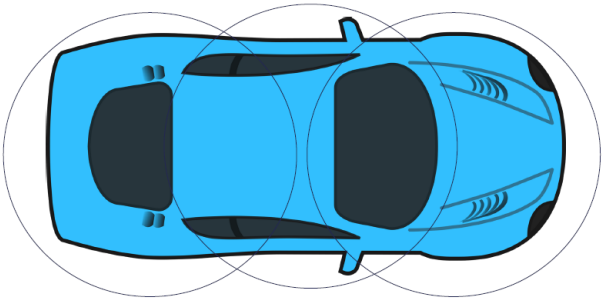


Fig. 1 three bounding box circles to account for obstacle detection.

To implement the traffic light detector, YOLOv2 network which was already adapted for the traffic light task was used.

The traffic light detection module was using the TrafficLightDetector class from the traffic_light_detector.py file that takes the camera' s data using camera_0, saves it as an image and then the traffic light detection module' s model makes the prediction using the predict function.. Possible output of the detect_traffic_light method are:

- 0, if the traffic light is green
- 1, if the traffic light is red

The behavioral_planner.py module was modified so that it would transition states based on the traffic light detection. In each state of the behavioral planner, the self_state method is called to check what is the traffic_light_state. Based on its state, the behavioral planner may change its state.

The rules of those transitions are the following:

1. If in FOLLOW LANE and there is a red light, transition to DECELERATE_TO_STOP which will eventually transition to STAY STOPPED
2. If in DECELERATE_TO_STOP and there is no traffic light or the traffic light is green, transition to FOLLOW LANE
3. If in STAY_STOPPED and there is no traffic light or the traffic light is green, transition to FOLLOW LANE

Steps of the traffic light detection are following:

1. Initialize the TrafficLightDetector object in main.py and pass it to the behavioral_planner.py.
2. Add a camera to carla settings.
3. Call the get_traffic_light_state method in the beahavioral_planner.py. Transition states based on the traffic light state if necessary.

After completing the two detection functions they had to be integrated into the behavioural planner. Both functions return a boolean of the collision and traffic light state, therefor the transitions in the function transition_state were to be adjusted to ensure that e.g. if the is_collision boolean is true that we shift from FOLLOWING LANE to DECELERATE TO STOP.

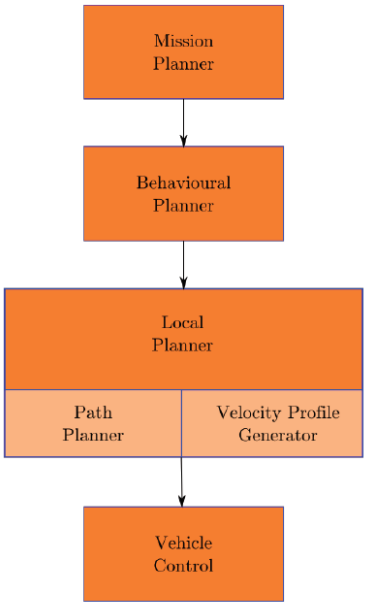


Fig. 2 Hierarchical planner and its modules

2 IMPLEMENTATIONS

In the following are all components of the architecture of the hierarchical planner described with particular reference to the behavioral planner, with high-level representation of all modules and their interactions. Fig. 2 shows each module and its level of abstraction.

2.1 Hierarchical planner

The hierarchical structure of motion planning ensures that the highly complex tasks of autonomous driving becomes more accessible and even more important, each module has its own level of abstraction according to their field of tasks. Hence the hierarchy starts from very high level planning to low level execution.

2.2 Mission Planner

The task of the mission planner is to plan the path from starting point to the destination. It is the first step in the hierarchical planner, since it only takes into account a high level of abstraction about the path. More detailed information about local behavior is not included. In most cases an A*-search is applied to find the shortest path.

2.3 Behavior planner

The behavior planner is a step down in abstraction and is already concerned with the local surroundings.

The general task of the behavior planner is to maintain safe and smooth transportation in a restricted environment with obstacles, street signs, traffic lights and lane boundaries,

information from sensors like RGB-cameras and LIDAR is evaluated and will account for the next state of the ego-vehicle.

The most intuitive and simple approach of behavioural planning is the finite state machine. It receives information from the sensors and according to those changes its current state or remains in the same state until the next iteration. In the traditional finite state machine, there are the three different states, "Follow lane", that means there is no obstacle ahead and the ego vehicle should continue the path towards the destination, "Decelerate to stop" which implies that a potential collision is ahead and the vehicle should slow down and lastly the state "Stay stopped" when the vehicle has reached a velocity of less than 1m/s and the ego vehicle should remain stopped due to a potential collision ahead or any street sign indication including red traffic lights. The current state of the behavior planner influences the local planner whether to keep the velocity, accelerate, brake or stay stopped.

2.4 Local Planner

The local planner receives the information of the current state of the car and has to choose a collision free and safe path in local scenarios. It is decomposed into a path planner and velocity profile generation.

The path planner is concerned with the lateral control and takes all possible paths into consideration, from which he chooses the safest, fastest and most comfortable.

Velocity profile generation on the other hand is concerned with the longitudinal movement and according to the state of the behavioral planner it will decelerate or accelerate or stay stopped.

2.5 Vehicle Control

All aforementioned abstract planning modules will then lead to an actuation of to maneuver the vehicle. The Vehicle Control module sends motion signals that are to be executed and receives the feedback of the actual movement. The ego vehicle is represented in the world as its x, y and z position furthermore we implemented its pitch roll and yaw on the world, after this we set its steer, throttle and brake' s limitations.

2.6 Traffic Light Detection

For the traffic light detection we were provided with a pretrained model from GitHub (link: https://github.com/affinis-lab/traffic-light-detection-module). Firstly we had to add a sensor which was an RGB-camera, bound forward, that perceived coloured images from the current scenery. In order to detect the current traffic light state, these images had to be saved and then evaluated by the traffic light module.

2.7 Obstacles Collision Control

The module in which we managed the obstacles collision is in the obstacle_detection module; "next_position" method predicts where will the agent be after N steps of simulation, "circles_for_detection" method get the orientation of the ego-vehicle, "check_for_obs" method get circles_centers, get obstacle data (included current and next position) and check whether obstacle location is in distance of radius.

3 ANALYSIS OF THE SYSTEM

3.1 Quantitative Analysis of the Results

We have run 10 significant tests out of which 90 % were successful. We documented 5 of them, since situations coincided.

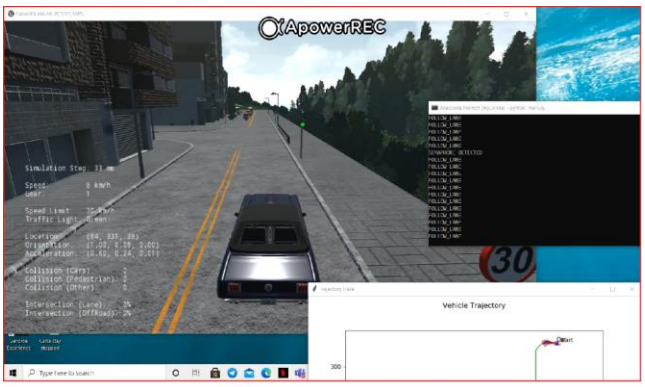
3.1 Qualitative Analysis of the Results

In the following google drive link there are contained three videos of the ego-car handling an intersection, a traffic light and avoiding vehicle as well as pedestrian collision:

https://drive.google.com/drive/u/1/folders/0AAYZWpwOx4-4Uk9PVA

Test 1

The ego rides smoothly through the intersection after changing the state from STAY_STOPPED to FOLLOW_LANE



Test 2

The ego starts decelerating when it detects obstacles around it.

Test 3

The ego-car slows down when detecting a traffic light and stays in STAY STOPPED state until it turns green.

Test 4

As well in a dark rainy weather scenario the ego-vehicle is able to detect the traffic light and decelerates in front of it.

Test 5

The following test took place in a late afternoon sunset scenario where light reflection is already changed and a sensor might miss an object. Nevertheless, the ego-vehicle decelerates when detecting obstacles upfront. One possible flaw that leaves room for improvement is that the ego stops earlier than it has to.

When the traffic light turns green and the other vehicles start acceleration the ego-vehicle transitions back to FOLLOW LANE without completely stopping. So that the journey can smoothly continue.

4 Conclusion

The tests showed a significant capability of the self-driving algorithm which can be considered as a baseline on which there is still room for improvement to ensure comfortable and safe driving.

One recurring problem was due to the implementation of waypoints in CARLA, where oftentimes the ego vehicle would stop quite early or quite late since it would try to reach the next waypoint rather than just stopping. Furthermore it sometimes took one of the trajectory lines that were quite far off and would therefore leave the lane boundaries.

5 References

1. Documentation about Carla "https://carla.readthedocs.io/en/latest/python_api/ "
2. Traffic Light Detection Modules "https://github.com/affinis-lab/traffic-light-detection-module"
3. Course material UNISA autonomous vehicle driving "https://elearning.unisa.it/course/view.php?id=2980 "