UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE ED ELETTRICA E MATEMATICA APPLICATA

Corso di Laurea Magistrale in Ingegneria Informatica

Autonomous Vehicle Driving Report

Group 6

Authors:

Antonio LANGELLA
Mat. 0622702011
a.langella31@studenti.unisa.it

Salvatore PAOLINO
Mat. 0622702016
s.paolino6@studenti.unisa.it

Michele MARSICO
Mat. 0622702012
m.marsico10@studenti.unisa.it

Adamo ZITO
Mat. 0622702026
a.zito32@studenti.unisa.it

ANNO ACCADEMICO 2023/2024

CONTENTS

INTRODUCTION

This report is the result of the final project of the course of Autonomous Vehicle Driving. The objective of the project is to develop an autonomous driving agent which maximizes the mean driving score on a set of two given routes. The driving score on a given route is the product between the route completion and the infraction penalty:

- **ROUTE COMPLETION**: It represents the percentage of the route distance completed by an agent;
- **INFRACTION PENALTY**: It is the product of the infractions committed. The base score is 1.0, and it is reduced at each infraction.

The average of each evaluation metric is computed when all scenarios in a route have been completed. These metrics are collectively referred to as "global metrics", with the global driving score being the main metric.
The agent is required to consider several different scenarios, including:

- Lane merging.
- Lane changing.
- Negotiations at traffic intersections.
- Handling traffic lights and traffic signs.
- Yielding to emergency vehicles.
- Coping with pedestrians, cyclists, and other elements.

In order to develop the agent, a baseline code was provided. The baseline code is a simple agent that can drive on the given routes, but it is not optimized to achieve the highest scores. Thus, secondary aim of the project is to analyze the provided baseline agent to identify its weaknesses and find the most critical points that need improvement to achieve the highest score. Then, by implementing new features and refactoring the existing code to achieve, we strive to achieve the highest possible scores on the given routes. The analysis will involve:

1. Identifying weaknesses in the baseline code;
2. Finding the critical points where improvements are needed to maximize scores;
3. Implementing strategies to address these weaknesses and improve overall performance.

Finally, a critical performance evaluation of the proposed agent will be conducted, comparing the results obtained with the baseline agent and highlighting both the strengths and weaknesses of the proposed agent.

CHAPTER 1

BASELINE ANALYSIS

This chapter is dedicated to the baseline analysis, which has been conducted via a preliminary analysis of the provided codebase followed by an assessment of the performances of the provided agent in the given scenarios. The aim here is to highlight the limitations of the baseline agent that arose during the analysis of each scenario. Finally, we conclude with a summary of the main issues of the agent which will be key to the aim of improving its performance.

1.1 Code analysis

Before the evaluation of the agent's performances on the given scenarios we thoroughly analyzed the provided code. It includes two carla agents named *basic_agent* and *behavior_agent*. As the second one is a strict extension of the first one, we will directly focus on the *behavior_agent*. In the baseline code the following modules can be easily identified:

1. A global planner;
2. A really simple local planner;
3. The longitudinal and lateral controllers.

Comparing the provided baseline with the Autoware architecture shows that several critical modules are missing, such as the localization, map, sensing and perception modules. This is because much of the code that should be handled by the missing modules is tightly coupled with the present modules as well as the agent class itself. The most noticeable missing modules is the behavioral planner, which is responsible for deciding the high level actions that the vehicle should take, such as lane changes, stops, etc. This logic is currently implemented in the *behavior_agent* class, and it is tightly coupled with the rest of the code. For these reasons, one of the aims of this project will be to better organize the code and separate the different modules, so that they can be easily replaced or extended.

1.1.1 Global Planner

The global planner's aim is to generate a high level route that the ego vehicle should follow, starting from the provided map of the road system we want to navigate in, as well as the start and goal positions on the map. This module works by extracting the topology of the OpenDRIVE map and generate an internal representation of the road system as a directed graph. Finally, the A* algorithm is applied on the said graph, using the distance as the cost to minimize.
Differently from the Autoware mission planner, the output of this module is a sequence of waypoints that the ego vehicle should follow, rather than a high level route. The main difference is that the global planner is already taking into account which lane the vehicle should be in at each waypoint, while Autoware's mission planner only provides a high level route that the vehicle should follow and the lane is chosen by the lower levels of the planning pipeline. This is a simplification was considered to be acceptable for the scope of the project, as no lane changes are required in the provided scenarios.

1.1.2 Local Planner

The provided local planner is very simple. It has two main features: it can follow a sequence of waypoints provided by the global planner, and it can generate new random waypoints on request. As the second feature is not required for the provided scenarios, we will focus on the first one. When a global plan is set, the local planner sets the reference for the lateral Stanley controller as the whole global plan. It will be the responsibility of the lateral controller to check which is the nearest waypoint of the global plan to the vehicle and set it as the current reference. During the execution of the local planner, starting from the global plan, the waypoints that are too near to the vehicle are pruned. Then, if no waypoints are left, the local planner will send an emergency stop command to the vehicle, otherwise it will pass the current target speed (set by the agent code) to the longitudinal PID controller.
The main limitation of the local planner is that it doesn't take into account collision avoidance with other vehicles or pedestrians. This is because, in the provided baseline, the agent code is responsible for detecting obstacles and deciding the high level actions that the vehicle should take. This is a limitation that we will address in the proposed solution.

1.1.3 Controllers

For the lateral control a Stanley controller is used while for the longitudinal control a PID controller is used. For the integral action the PID stores the last 10 errors and computes their sum. This is clearly a limitation of the PID controller that we will address in the proposed solution.

1.1.4 Agent behavior

As we said the high level planning logic (that would normally be embedded in a behavioral planner) is implemented in the agent class itself. Even if the logic of the baseline agent isn't well-structured, the agent's behavior can be traced back to a Moore finite-state machine with the following states:

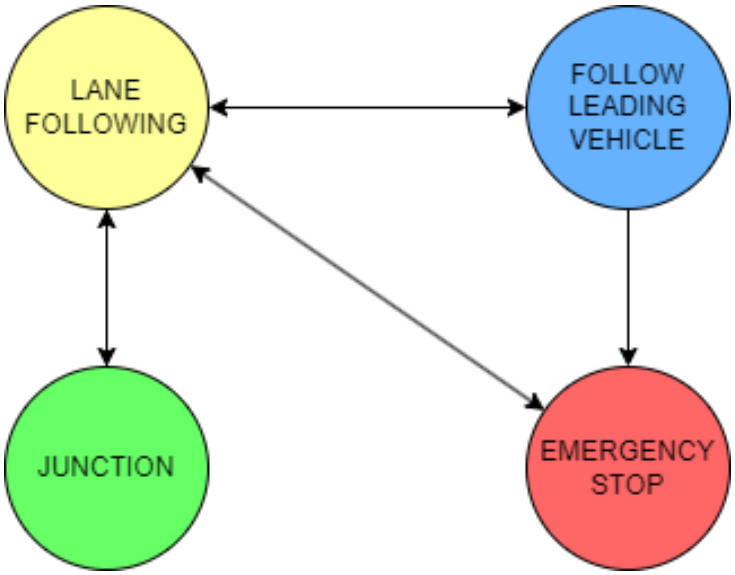


Figure 1.1: Finite-state machine of the baseline agent. Transition conditions and actions are omitted for simplicity.

1. *Lane following*: the agent tries to follow the waypoints provided by the local planner, setting the target speed to the maximum speed allowed by the road minus a safety margin;

2. *Follow leading vehicle*: the agent tries to follow the leading vehicle, setting the target speed to the speed of the leading vehicle. If the leading vehicle is too far away the agent accelerates while if the leading vehicle is too near the agent decelerates. This state is triggered when a vehicle is detected in front of the ego vehicle and the distance is above a certain threshold;

3. *Junction*: the agent tries to follow the waypoints provided by the local planner, but it sets a lower target speed than in the lane following state;

4. *Emergency stop*: the agent stops the vehicle violently, without taking into account the vehicle dynamics. This state is triggered when a vehicle or a pedestrian is detected in front of the ego vehicle and the distance is below a certain threshold. This state is also triggered when the vehicle is under the influence of a red traffic light.

Fig. 1.1 shows the finite-state machine that describes the agent's behavior. We also want to emphasize the fact that, for the sake of representation, state transitions and actions have been omitted, but only states are present.

1.2 Performance evaluation on the given scenarios

The first thing that was done, to facilitate the analysis, was to extract each scenario from the provided routes, creating a new route for each scenario. For each subroute, the starting waypoint is the trigger waypoint of the previous scenario and the ending waypoint is the trigger of the next scenario. This is accomplished by the `generate_subroutes.py` script in the `utils` folder (Fig. 1.2). Notice that the script also changes time of day to have a better visibility in route 1.

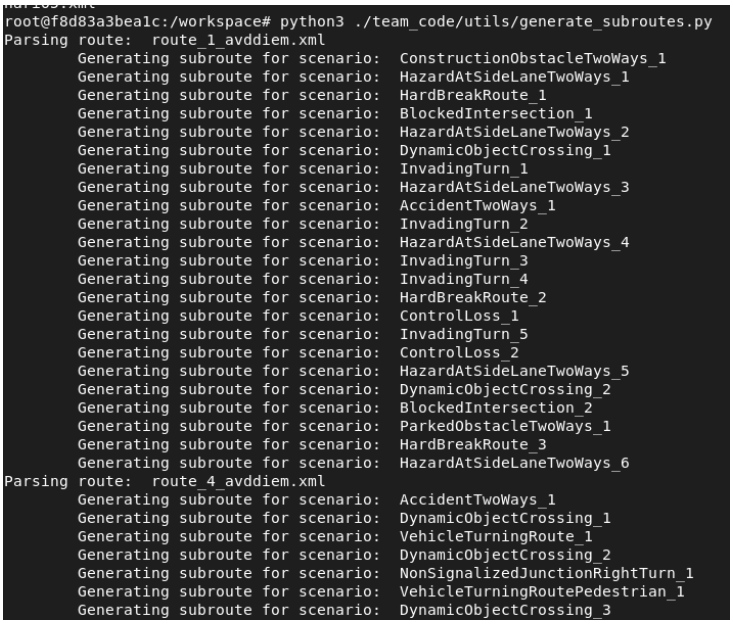


Figure 1.2: Output of the `generate_subroutes.py` script.

1.2.1 Route 1

The first route provided consists of a total of 23 scenarios, some of them of the same type. It takes place on a suburban road with wet asphalt, while it is raining, at night. A morning time has been set to simplify visualization.

Scenario 0

The scenario is of type `ConstructionObstacleTwoWays`, so the agent has to pass a roadwork signal, encroaching on the opposite lane. The agent does not recognize the obstacle and crashes into it. As a result, the simulation ends after a timeout. (Figs. 1.3 and 1.4)

Scenario 1

This scenario is of type `HazardAtSideLaneTwoWays`, in which the agent has to overtake a group of cyclists, by invading the opposite lane of traffic. In this case, the agent does not overtake, but stays in the rear of the cyclists, and in case he gets too close, he enters the emergency braking state. When the cyclists stop, the agent stops behind them, so the scenario ends with the timeout. (Figs. 1.5 and 1.6)

Scenario 2

The scenario is of type `HardBreakRoute`, in which all the cars in front of the agent brake to a stop, and then start again. Agent stops and then restarts correctly. The only penalty in this scenario is `min_speed_infractions` in which the agent's speed is less than that of the other cars in the scenario. (Fig. 1.7)

Figure 1.3: Scenario 0 - map

Figure 1.4: Scenario 0 - camera

Figure 1.5: Scenario 1 - map

Figure 1.6: Scenario 1 - camera

Figure 1.7: Scenario 2 - map

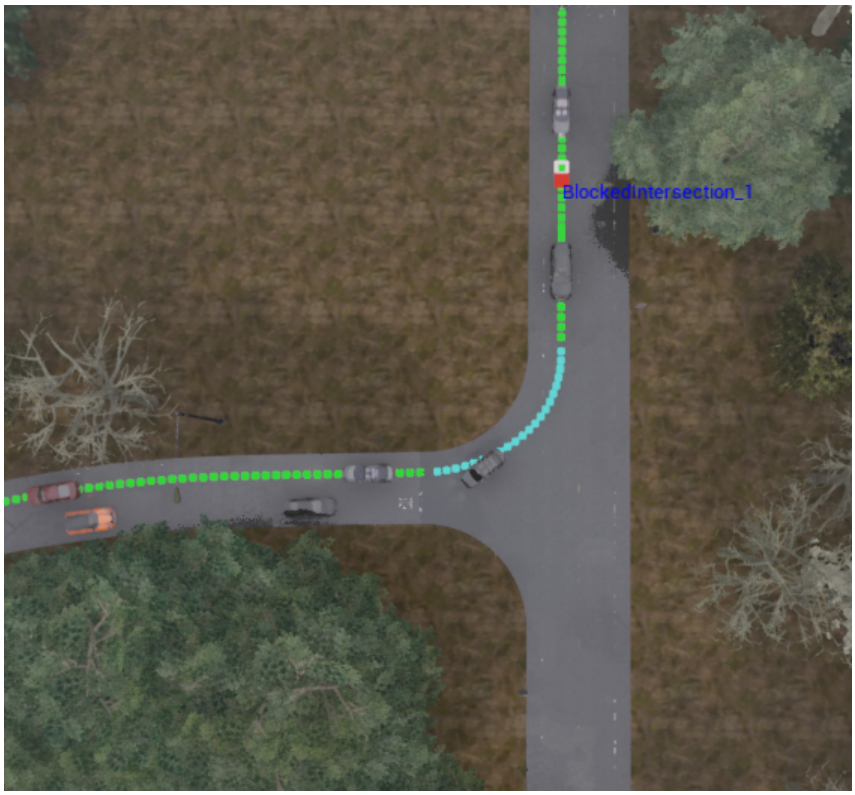


Figure 1.8: Scenario 3 - map

Scenario 3

This scenario is of the `BlockedIntersection` type, in which the agent makes a turn and finds the road busy. In this case, the car stops properly when it encounters traffic, and then starts again normally. Here also the only penalty in this scenario is `min_speed_infractions`. (Figs. 1.8 and 1.9)

Scenario 4

This scenario is of the type `HazardAtSideLaneTwoWays`, like scenario 1, but unlike the latter, the agent, when the cyclists have stopped, overtakes them by getting very close, and in some

Figure 1.9: Scenario 3 - camera

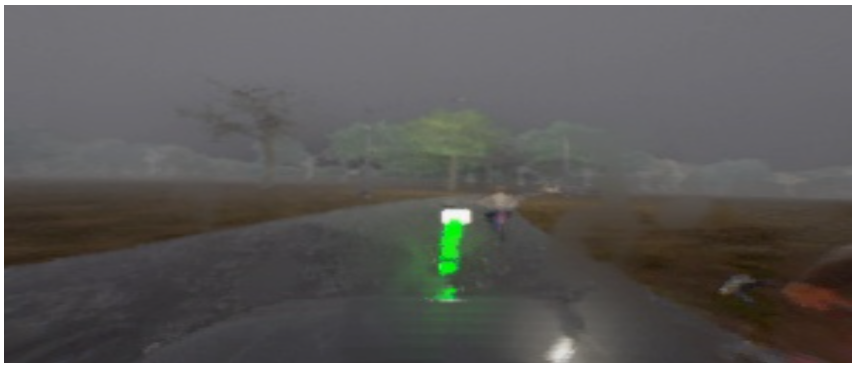


Figure 1.10: Scenario 4 - camera

cases hits them (Fig. 1.10).

Scenario 5

This scenario is of type `DynamicObjectCrossing`, in which a pedestrian crosses the street by coming up from behind a car. The agent does not recognize the pedestrian and collides with it.

Scenario 6

This scenario is of the `InvadingTurn` type, in which vehicles in the opposing lane partially encroach on the agent's lane due to road work. The agent should give them space by moving slightly off-center, but the baseline agent doesn't react to the encroachment. Nonetheless, the agent correctly passes the scenario with a `min_speed_infractions` penalty. (Figs. 1.11 and 1.12)

Scenario 7

This scenario is also of the type `HazardAtSideLaneTwoWays`, the agent behaves like in scenarios 1 and 4.

Scenario 8

The scenario is of type `AccidentTwoWays`, so the vehicle finds an accident in its path and is forced to change lanes by invading the opposite lane. In this scenario the agent provided does not perform the task at all, in fact he does not overtake, but goes to hit the police car and gets stuck, so the scenario ends after a timeout. (Figs. 1.13 and 1.14)

Scenario 9

This scenario is of the `InvadingTurn` type, and the agent behaves exactly like scenario 6. (Fig. 1.15)

Scenario 10

This scenario is also of the `HazardAtSideLaneTwoWays` type, and the ego car behaves just like the previous ones.

Figure 1.11: Scenario 6 - map

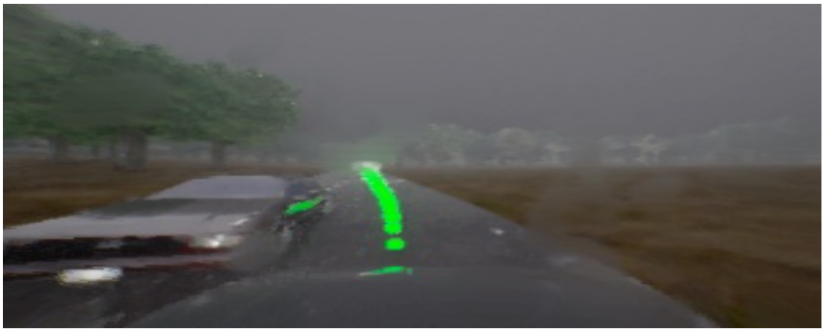


Figure 1.12: Scenario 6 - camera

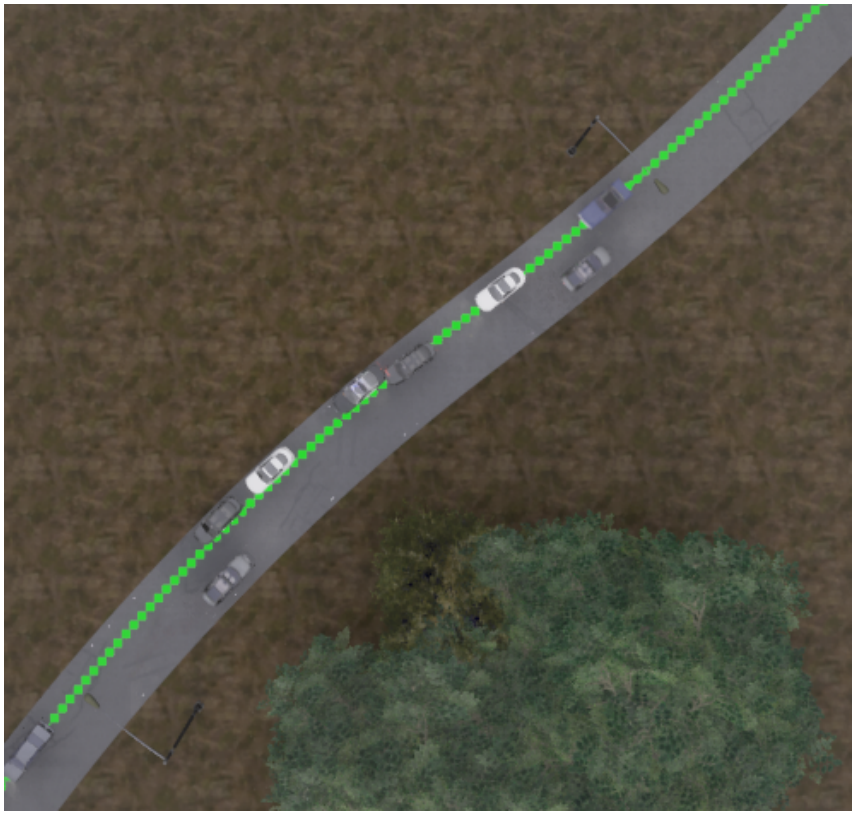


Figure 1.13: Scenario 8 - map

Figure 1.14: Scenario 8 - camera

Figure 1.15: Scenario 9 - map

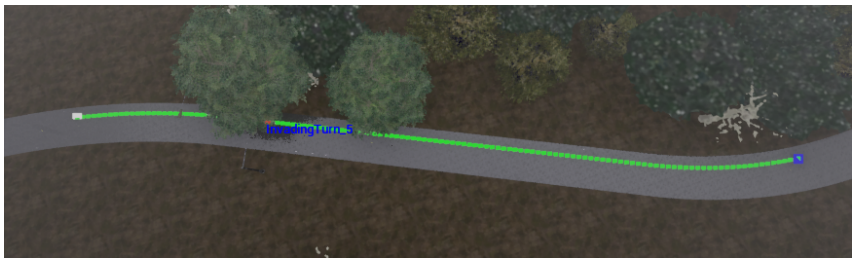


Figure 1.16: Scenario 15 - map

Scenario 11

This scenario is of the `InvadingTurn` type, and the agent behaves just like the previous ones.

Scenario 12

The scenario is of the `InvadingTurn` type, so the ego car behaves exactly like the previous scenarios of the same type, and the only penalty is still `min_speed_infractions`.

Scenario 13

The scenario is of type `HardBreakRoute` as in scenario 2, and the agent behaves exactly like that scenario.

Scenario 14

The scenario is of the `ControlLoss`, where the vehicle is tested against to bad road conditions. The agent behaves correctly and the scenario ends with a success.

Scenario 15

This scenario is of the `InvadingTurn` type, and the agent behaves just like the previous ones, with the only penalty being `min_speed_infractions`. (Fig. 1.16)

Scenario 16

This scenario is of the `ControlLoss` type, and the vehicle behaves just like the scenario 14.

Scenario 17

The scenario is of the `HazardAtSideLaneTwoWays` type, and also here the agent needs to overtake a group of cyclists, but as in the previous scenarios of the same type, the agent does not overtake, but stays behind the cyclists, and if the cyclists stop, the agent stops behind them, so the scenario ends with the timeout. (Figs. 1.17 and 1.18)

Scenario 18

This scenario is of the `DynamicObjectCrossing` type, where a pedestrian crosses the street, so the agent has to stop and wait for the pedestrian to cross. The agent behaves correctly

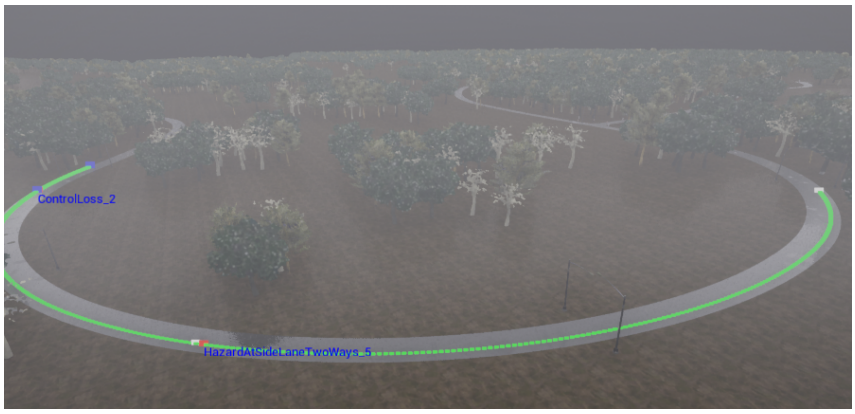


Figure 1.17: Scenario 17 - map

Figure 1.18: Scenario 17 - camera

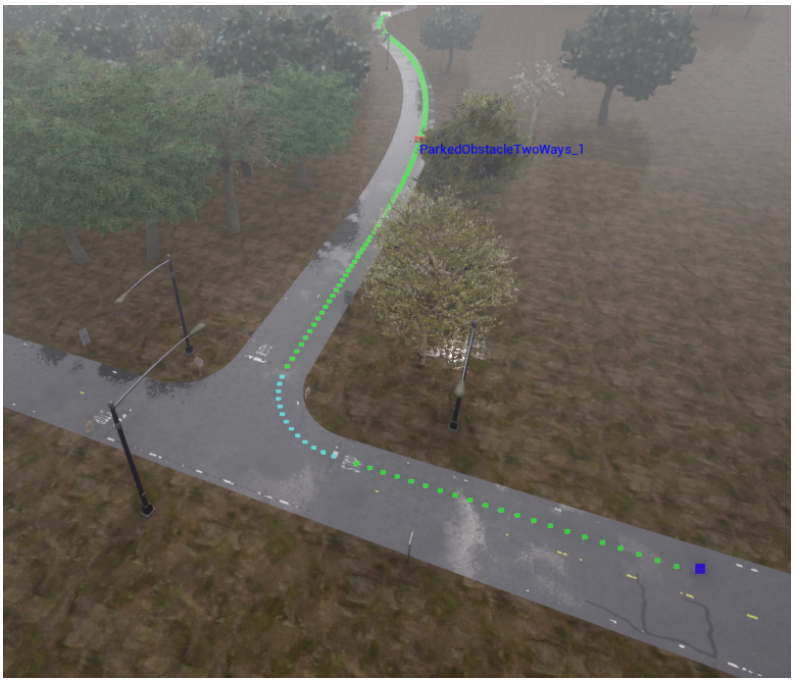


Figure 1.19: Scenario 20 - map

and the scenario ends with a success. Even here the only penalty in this scenario is `min_speed_infractions`.

Scenario 19

This scenario is of the `BlockedIntersection` type, and the agent behaves just like the scenario 3. In this case the agent stops correctly when he encounters traffic, and then starts again normally, but there is the `min_speed_infractions` penalty and the `running_stop` penalty, because the agent does not respect a stop sign.

Scenario 20

This scenario is of the `ParkedObstacleTwoWays` type, in which the agent has to pass a parked car by encroaching on the opposite lane. In this scenario, the agent has to make a turn at an intersection, but he disregards the stop sign, and when he encounters the parked car, he collides with it. Therefore, the simulation ends due to the expiration of the timeout. (Figs. 1.19, 1.20 and 1.21)

Scenario 21

This scenario is also `HardBreakRoute` type. And again the main problem here involves crossing an intersection, and colliding with another car that is crossing the intersection at the same time. Because of this, the simulation ends with a timeout. (Figs. 1.22, 1.24 and 1.23)

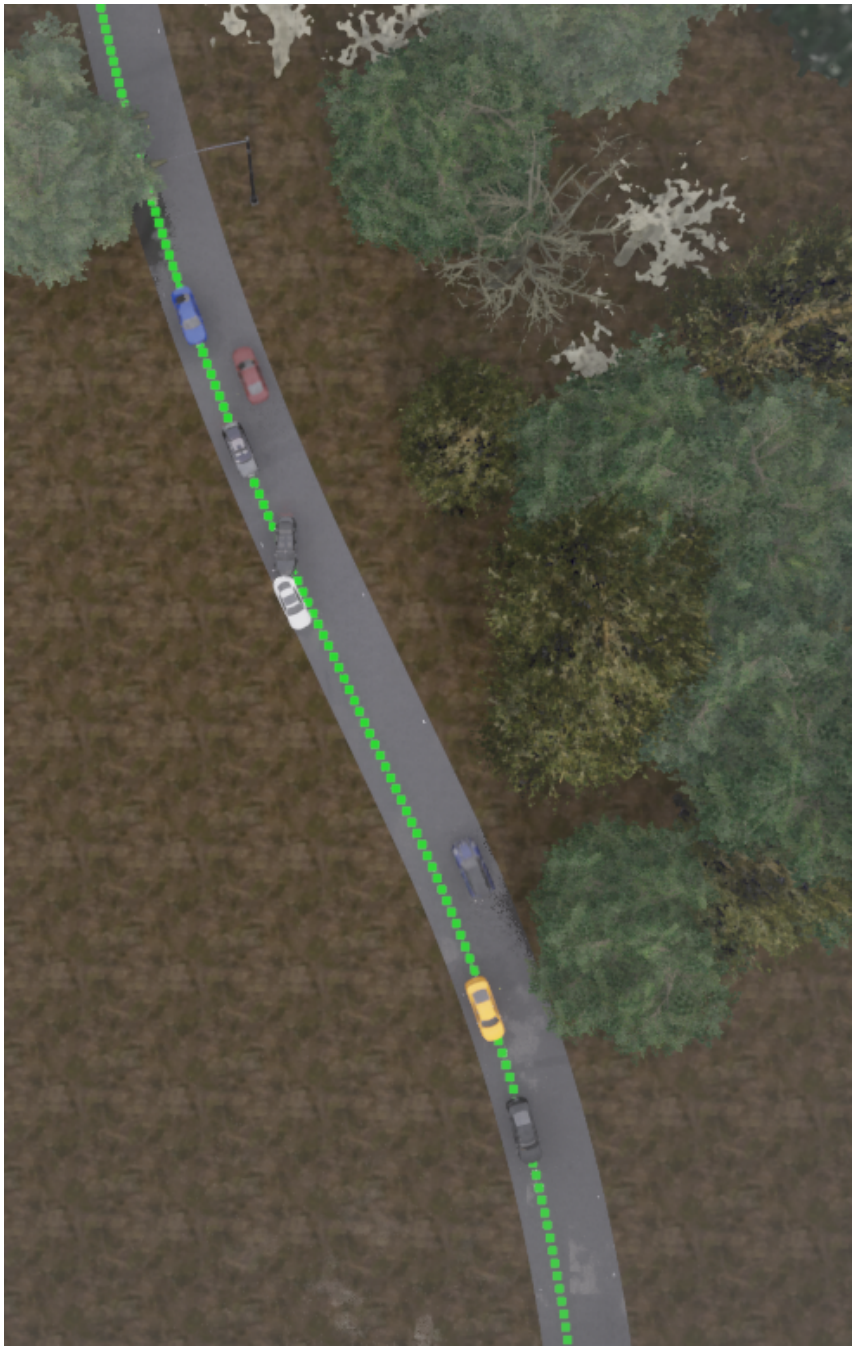


Figure 1.20: Scenario 20 - map2

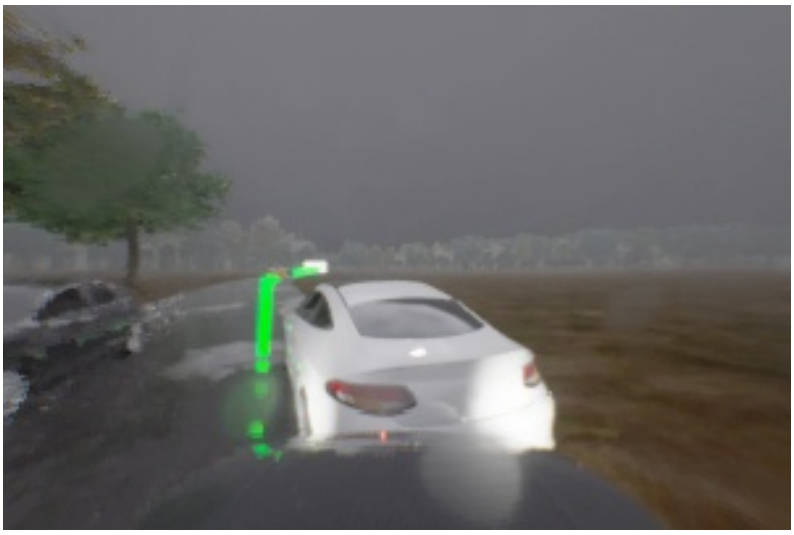


Figure 1.21: Scenario 20 - camera

Figure 1.22: Scenario 21 - map

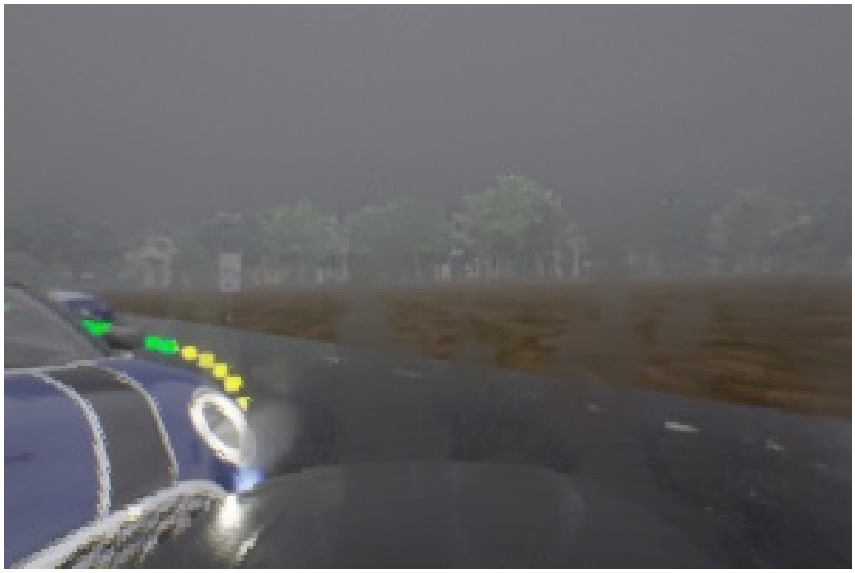


Figure 1.23: Scenario 21 - camera

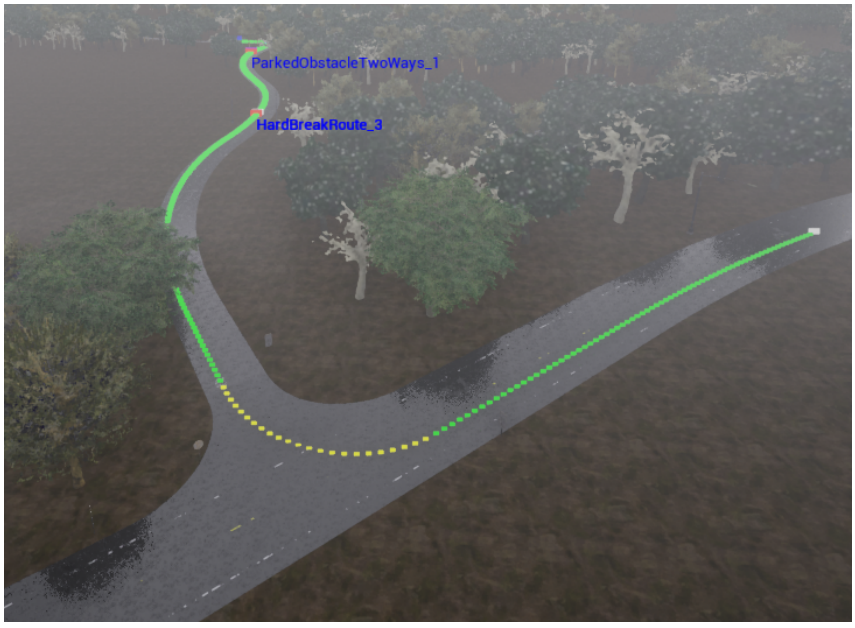


Figure 1.24: Scenario 21 - map2

Figure 1.25: Scenario 22 - map

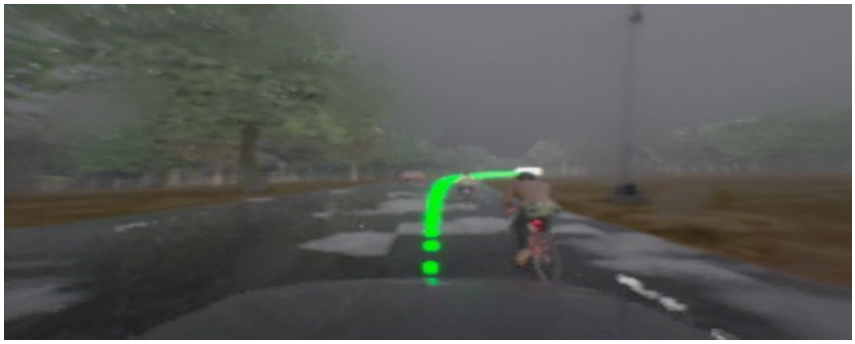


Figure 1.26: Scenario 22 - camera

Scenario 22

This scenario is also of type `HazardAtSideLaneTwoWays`, so the agent has to overtake a group of cyclists, going to invade the opposite lane, but he does not do so. Again there is an intersection to cross and the stop sign is not obeyed, and the car collides with a car that is crossing the intersection, and as a result the simulation ends with a timeout. (Figs. 1.25 and 1.26)

Total score

The route score is the percentage of route completion achieved by the agent. In this case, the route score is:

$$\text{score\_route} = 12.14$$

The total penalties are the result of the following infractions:

- Collision with a static object (static.prop.trafficwarning)
- Scenario timeout
- Route timeout

Each infraction contributes to the total penalty score. The infractions recorded are:

- Number of collisions with layout objects: 1
- Number of scenario timeouts: 1
- Number of route timeouts: 1

The total penalty score is calculated by adding the penalties for each type of infraction. In this case:

$$\text{score\_penalty} = 0.455$$

The composed score is obtained by multiplying the route score with the penalty score:

$$\text{score\_composed} = \text{score\_route} \times \text{score\_penalty}$$

Substituting the values:

$$\text{score\_composed} = 12.14 \times 0.455 = 5.5237$$

Thus, the composed score reflects the route completion percentage adjusted for the penalties incurred during the scenario. The fundamental issue that occurs in this route is the route timeout, which prevents the completion of the route.

1.2.2 Route 4

The second route provided consists of 7 scenarios. It takes place in a country town with many intersections without traffic lights, but with stop signs.

Scenario 0

The scenario is of type `AccidentTwoWays`, as in scenario 7 of route 1. And here, too, the agent behaves in the same way. (Figs. 1.27 and 1.28)

Scenario 1

The scenario is of type `DynamicObjectCrossing`, in which a pedestrian crosses in front of the vehicle. As the pedestrian passes, the agent correctly avoids the pedestrian by entering the emergency braking state and waits for the pedestrian to pass, then starts again. But the agent still commits an infraction as he doesn't obey a stop sign at an intersection. Furthermore, as for other previous scenarios, there was a penalty when the vehicle traveled too slow compared to other surrounding cars. (Fig. 1.29)

Figure 1.27: Scenario 0 - map

Figure 1.28: Scenario 0 - camera

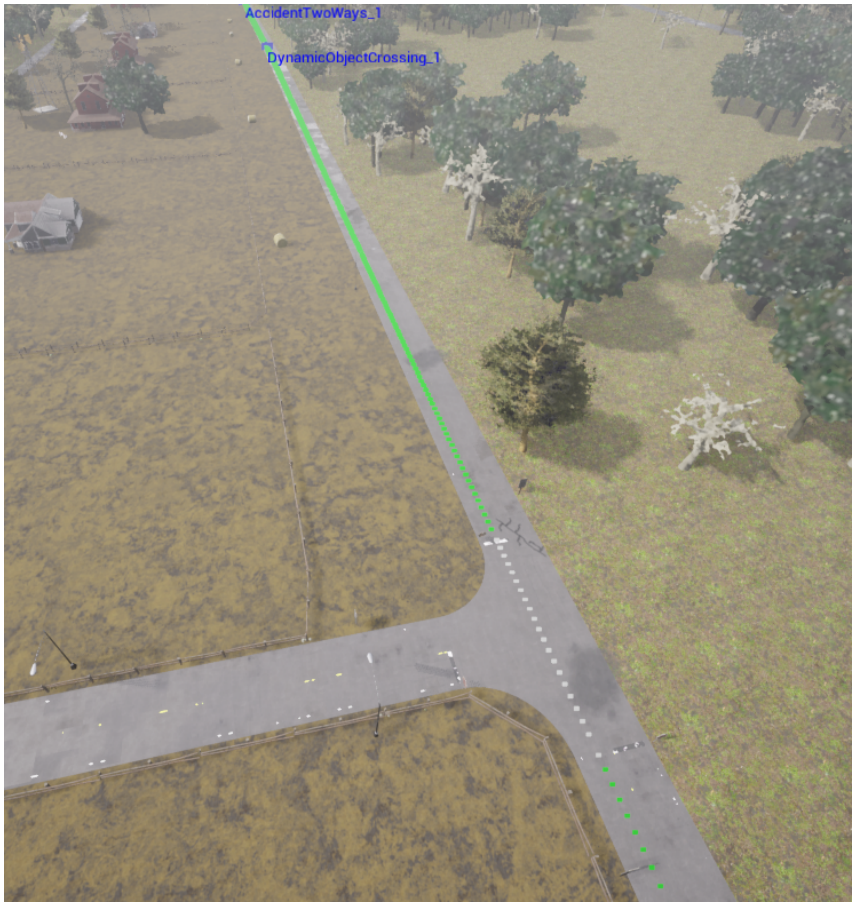


Figure 1.29: Scenario 1 - map

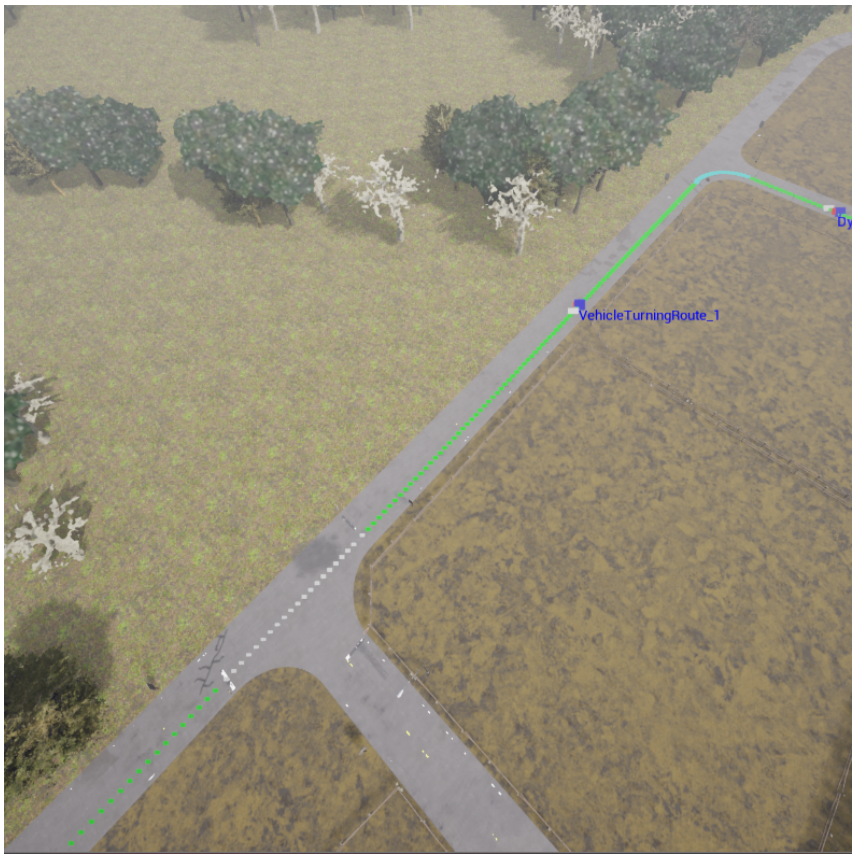


Figure 1.30: Scenario 2 - map

Scenario 2

The scenario is of type `VehicleTurningRoute`, so there is a bicycle that has to cross the road when the ego vehicle has to turn. In this case the agent doesn't recognize the bicycle and collides with it. Also, the agent has to cross two intersections, both having a stop sign, and in both cases the agent does not stop, and commits an infraction. (Fig. 1.30)

Scenario 3

This scenario is also of type `DynamicObjectCrossing`. The car completes the scenario correctly, but are given a penalty for the infraction regarding stop sign, and one for when the vehicle is traveling too slow relative to other cars.

Scenario 4

The scenario type is `NonSignalizedJunctionRightTurn`, in which the car has to turn right at an intersection where there is no traffic light. The penalty is always for not stopping at the stop sign and going too slowly compared to other cars.

Scenario 5

The scenario is of type `VehicleTurningRoutePedestrian`, in which the car has to turn at an intersection and there is a pedestrian crossing the intersection. In this case scenario the agent,

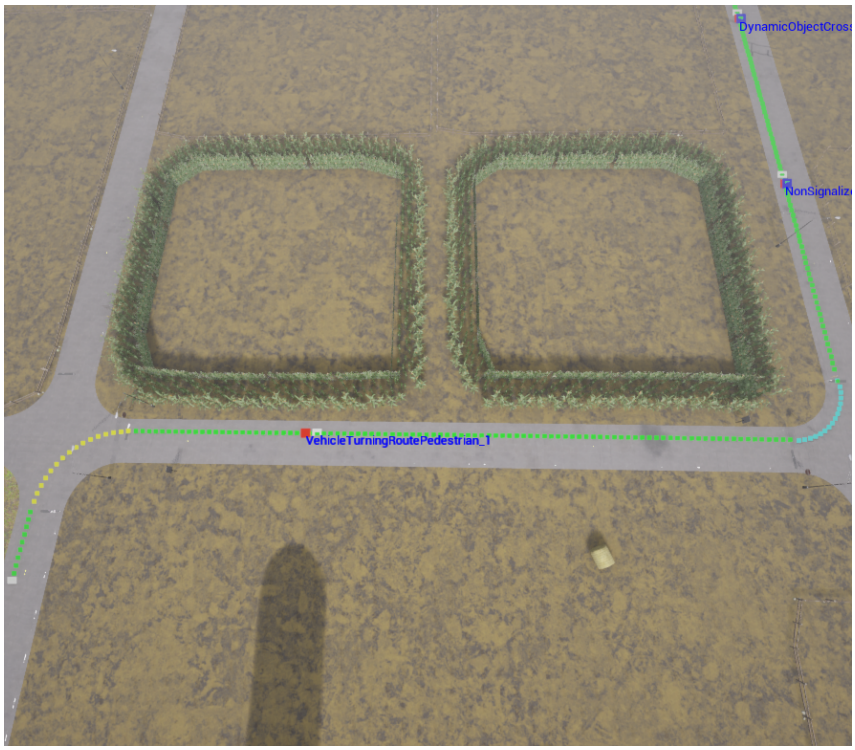


Figure 1.31: Scenario 5 - map

as seen earlier, does not stop at the two stop signs present, and runs over the pedestrian. In addition, the minimum speed penalty is also present here. (Fig. 1.31)

Scenario 6

This scenario is of the `DynamicObjectCrossing` type, in which a pedestrian suddenly crosses coming out from behind a bale of hay. The agent behaves correctly with the pedestrian, but fails to obey the stop sign. (Fig. 1.32)

Total score

The route score is the percentage of route completion achieved by the agent. In this case, the route score is:

$$\text{score\_route} = 100.0$$

In this scenario, the agent completed the route with a score of 100.0 for route completion. However, the composed score is significantly reduced due to the penalties incurred. The penalties incurred in this scenario are:

- Collisions with pedestrians
- Collisions with vehicles
- Running stop signs
- Scenario timeout

The total penalties result from the following infractions:

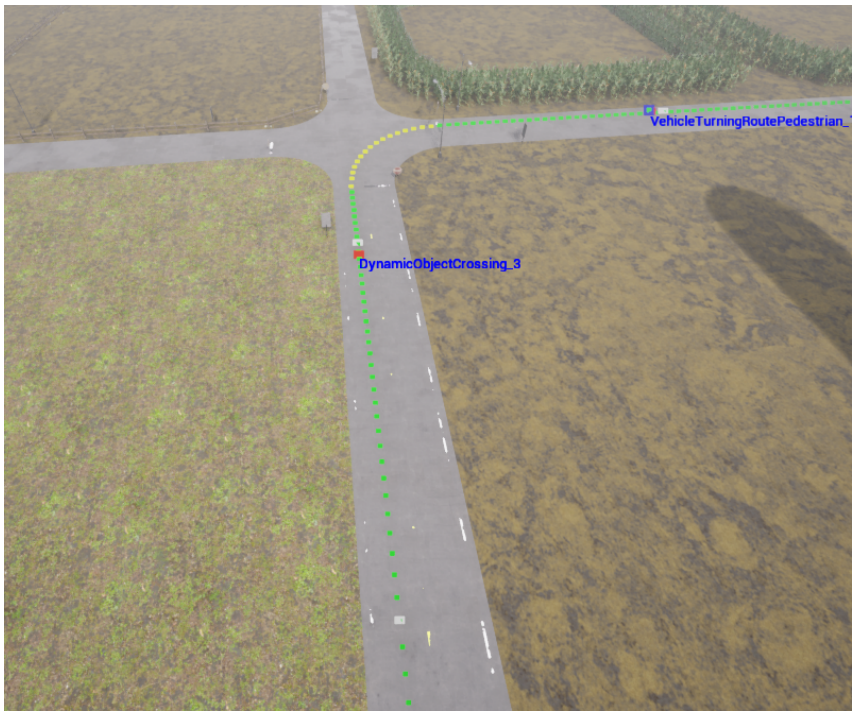


Figure 1.32: Scenario 6 - map

- Collisions with pedestrians: 2
- Collisions with vehicles: 3
- Running stop signs: 3
- Scenario timeout: 1

The total penalty score is calculated by adding the penalties for each type of infraction. In this case:

$$\text{score\_penalty} = 0.019354$$

The composed score is obtained by multiplying the route score with the penalty score:

$$\text{score\_composed} = \text{score\_route} \times \text{score\_penalty}$$

Substituting the values:

$$\text{score\_composed} = 100.0 \times 0.019354 = 1.93536$$

Thus, although the agent completed the route, the significant number of infractions, especially collisions and running stop signs, greatly reduced the final composed score. This indicates that the agent's performance in terms of safety and adherence to traffic rules needs improvement, even though the route was completed.

1.3 Conclusions

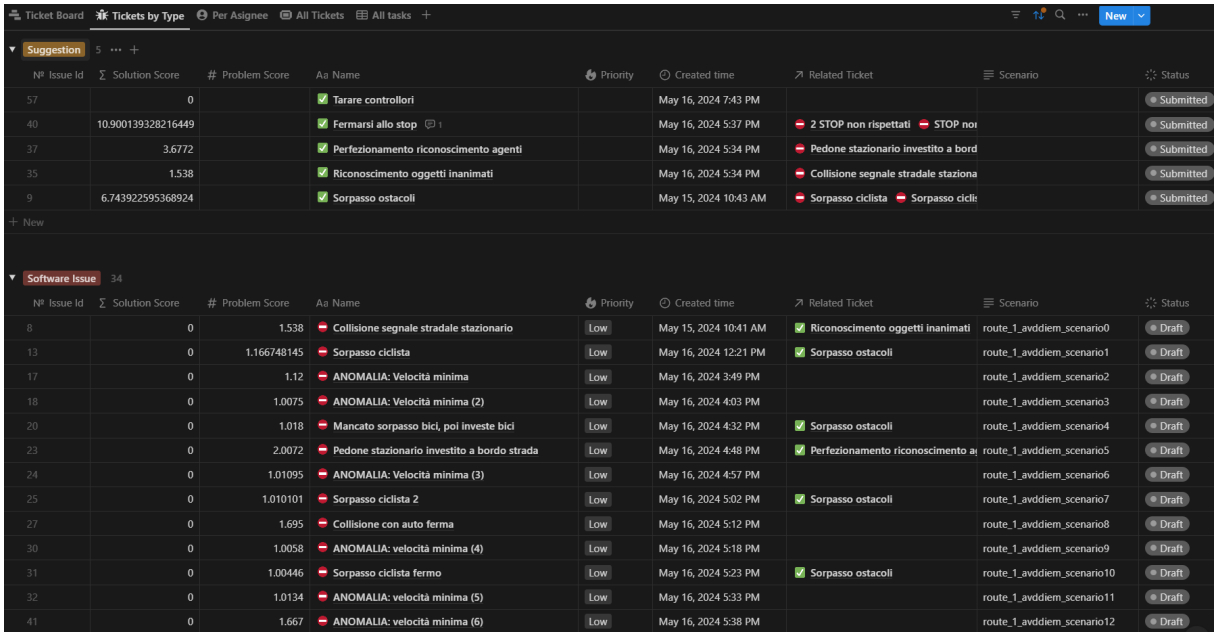
Following the analysis of all the scenarios, we want to summarize the main issues of the provided agent:

- The agent does not respect the stop signs, crossing the intersections without stopping;
- The agent does not respect the speed limits, often traveling at a speed lower than the minimum allowed or lower than the surrounding traffic;
- The agent does not overtake still vehicles or static obstacles ahead, but stops behind them;
- The agent does not overtake the cyclists when it is possible to do so;
- The agent does not recognize the roadwork signs or other objects on the side of the road, so it does not slow down or change lanes, but crushes into them;
- In the intersections, the agent does not respect the right of way, often causing accidents.

After this qualitative analysis we conducted a quantitative prioritization of the issues, in order to focus on the most critical ones. The prioritization was based on the frequency of the penalties and the severity of the infractions. To perform the analysis to the best of our ability, an issue tracking system was created (Fig. 1.33), in which a ticket was created for each individual penalty in each scenario. Each ticket was assigned a problem score calculated as the inverse of the sum of penalty scores due to that problem. Notice that the penalty due to route completion was not considered in this calculation. Then we proposed solutions to the given problems and associated them with the corresponding tickets. Finally, every solution was assigned a score as the sum of the scores of the tickets it solves. This gave us a quantitative measure of the importance of each problem and the effectiveness of the proposed solutions.

This analysis showed that, given the scenarios provided, the most present penalties are `running_stop`, `collisions_layout` and `collisions_vehicle`, which are the most common infractions committed by the agent. The most effective proposed solutions are

1. **Stop at stop signs**: the agent should stop at stop signs, waiting for the right of way;
2. **Overtaking**: the agent should be able to overtake still vehicles or static obstacles ahead, rather than stop behind them;
3. **Improve the collision avoidance system**: the agent should be able to recognize pedestrians and other vehicles in difficult situations, avoiding collisions;
4. **Recognize static obstacles**: the agent should be able to recognize roadwork signs or other objects on the side of the road, slowing down or changing lanes;



Ticket Board | Tickets by Type | Per Asignee | All Tickets | All tasks | New

Suggestion 5

Nº Issue Id	Solution Score	Problem Score	Name	Priority	Created time	Related Ticket	Scenario	Status
57	0		Tarare controllori		May 16, 2024 7:43 PM			Submitted
40	10.900139328216449		Fermarsi allo stop 1		May 16, 2024 5:37 PM	2 STOP non rispettati, STOP noi		Submitted
37	3.6772		Perfezionamento riconoscimento agenti		May 16, 2024 5:34 PM	Pedone stazionario investito a bord		Submitted
35	1.538		Riconoscimento oggetti inanimati		May 16, 2024 5:34 PM	Collisione segnale stradale stazionа		Submitted
9	6.743922595368924		Sorpasso ostacoli		May 15, 2024 10:43 AM	Sorpasso ciclista, Sorpasso cicli		Submitted

+ New

Software Issue 34

Nº Issue Id	Solution Score	Problem Score	Name	Priority	Created time	Related Ticket	Scenario	Status
8	0	1.538	Collisione segnale stradale stazionario	Low	May 15, 2024 10:41 AM	Riconoscimento oggetti inanimati	route_1_avddiem_scenario0	Draft
13	0	1.166748145	Sorpasso ciclista	Low	May 16, 2024 12:21 PM	Sorpasso ostacoli	route_1_avddiem_scenario1	Draft
17	0	1.12	ANOMALIA: Velocità minima	Low	May 16, 2024 3:49 PM		route_1_avddiem_scenario2	Draft
18	0	1.0075	ANOMALIA: Velocità minima (2)	Low	May 16, 2024 4:03 PM		route_1_avddiem_scenario3	Draft
20	0	1.018	Mancato sorpasso bici, poi investe bici	Low	May 16, 2024 4:32 PM	Sorpasso ostacoli	route_1_avddiem_scenario4	Draft
23	0	2.0072	Pedone stazionario investito a bordo strada	Low	May 16, 2024 4:48 PM	Perfezionamento riconoscimento a	route_1_avddiem_scenario5	Draft
24	0	1.01095	ANOMALIA: Velocità minima (3)	Low	May 16, 2024 4:57 PM		route_1_avddiem_scenario6	Draft
25	0	1.010101	Sorpasso ciclista 2	Low	May 16, 2024 5:02 PM	Sorpasso ostacoli	route_1_avddiem_scenario7	Draft
27	0	1.695	Collisione con auto ferma	Low	May 16, 2024 5:12 PM		route_1_avddiem_scenario8	Draft
30	0	1.0058	ANOMALIA: velocità minima (4)	Low	May 16, 2024 5:18 PM		route_1_avddiem_scenario9	Draft
31	0	1.00446	Sorpasso ciclista fermo	Low	May 16, 2024 5:23 PM	Sorpasso ostacoli	route_1_avddiem_scenario10	Draft
32	0	1.0134	ANOMALIA: velocità minima (5)	Low	May 16, 2024 5:33 PM		route_1_avddiem_scenario11	Draft
41	0	1.667	ANOMALIA: velocità minima (6)	Low	May 16, 2024 5:38 PM		route_1_avddiem_scenario12	Draft

Figure 1.33: Issue tracking system

CHAPTER 2

BASELINE IMPROVEMENTS

In this chapter we will describe the refactoring and the new features that we implemented in the baseline code. The refactoring was necessary to better organize the code and separate the different modules, so that they can be easily replaced or extended. The new features were implemented to address the limitations of the baseline code and to improve the agent's behavior.

2.1 Refactoring

Here is a non-exhaustive list of the main improvements we made to the agent's code:

- Our main improvement to the agent's code is to decouple the code in the behavioral planner. The first thing done was to formalize its logic by expliciting a finite-state machine. Specifically, we implemented a FSM that is close to MATLAB's Stateflow, in which each state has the `on_enter`, `action`, and `on_exit` functions to specify the behavior of the automaton upon entering a state, within a state, and exiting a state, respectively. We chose this hybrid form rather than a more strict Mealy or Moore machine to allow for greater flexibility in implementation. To achieve this, the *DrivingState* class was created to define a common interface for a driving state. Each driving state is a singleton. The driving state determines the next action to be executed by the agent based on the current state of the agent and the environment. Each state implements the `on_enter`, `action`, and `on_exit` methods. The `on_enter` method is called when the state is entered, the `action` method is called at every step of the simulation when the agent is in the state, and the `on_exit` method is called when the state is exited. Transitions between states are managed by the behavioral planner module, which determines the agent's next state based on its current state and environment, using a finite-state automaton to model the agent's behavior.
- We defined a *Singleton* class to manage objects shared across the entire application, such as the *Map* module, the *localization* module and the driving states.

- We developed a new *Waypoint* class to represent waypoints on the map, addressing the limitations of the original implementation in the CARLA PythonAPI. Those waypoints are limited in the fact that their transform is always tied to a road and has a fixed orientation. We exploited Python's concept of properties to create a more flexible waypoint class that can be used to represent waypoints on any surface, not just roads, and with any orientation. Thanks to the properties, the transformation of the waypoint is calculated on the fly starting from the transform of CARLA's waypoint, allowing for more dynamic and flexible waypoint management while keeping the same interface as the original CARLA waypoint to not break the existing code.

- The management of traffic lights was exported from the behavioral agent to the TrafficLightsManager class.

- A significant change involved the local planner, where we added a motion planner to handle obstacles and choose the locally optimal path. However, later we discarded the motion planner for a simpler and more effective approach (as we stated before).

- We enhanced the perception capabilities by integrating several detection modules: *ObstacleDetectionManager*, *PedestrianDetectionManager*, *TrafficLightDetectionManager*, and *VehicleDetectionManager*. These modules, togeteher with the new implemented ClusterDetectionManager and *RoadSignsDetectionManager*, ensure comprehensive detection of dynamic and static obstacles, pedestrians, traffic lights, and other vehicles, respectively. Previously, the business logic of these detection tasks was all integrated within the behavior agent, making the system less modular and harder to maintain. These refactoring activities have brought significant benefits. The decoupling of code has improved modularity, making the system easier to understand, maintain, and extend.

The formalization of the finite-state automaton has provided a clear structure for state management, enhancing the predictability and reliability of the system's behavior. The integration of detection modules has greatly improved the vehicle's perception capabilities, allowing for more accurate and timely decision-making. Overall, these improvements have resulted in a more robust and scalable autonomous driving system, capable of navigating complex environments with increased safety and efficiency.

2.1.1 Controllers problems

Regarding the controllers, we noticed a very serious error, which was that the PID longitudinal controller output was misinterpreted by carla. In fact a saturation between -1.0 and 1.0 was applied to the output of the PID and fed to the vehicle. This is wrong as the output of the PID must be first normalized in the range $[-1.0, 1.0]$ considering the vehicle's maximum acceleration, then fed to the carla vehicle. This meant that when the controller calculated a control input greater than 1, it was fixed to 1. This caused severe control effort and a very high jerk. In addition, the output was also interpreted with the wrong unit of measurement. To solve this problem, we scaled the controller output correctly.

Another problem with the PID controller is that it is subject to windup. This is a phenomenon that occurs when the controller is not able to reach the setpoint due to actuator

saturation, and the integral term accumulates an error that is not reset. This causes the controller to overshoot the setpoint and oscillate around it. To partially solve this problem we increased the size of the buffer of the PID controller, in which the past error is saved that is used for the integral action, and we used for the derivative action only the derivative of the reference. However a proper antiwindup strategy should still be implemented.

After that, we tuned both the longitudinal PID controller and the lateral Stanley controller using a manual tuning, as we don't have a model of the vehicle. This also prevented us from using a feedforward term in the controller.

2.2 Implemented features

In this paragraph we try to give a non-exhaustive view of the main implemented features. Most of the features are implemented in the behavioral planner, but some of them are implemented in the local planner are directly related to those in the behavioral planner.

2.2.1 Local Planner

In order to implement most of the features in the behavioral planner we needed to extend the features of the local planner. A first change we have made to the local planner is that there is now a distinction between the global plan and the local plan, which now includes only a window of waypoints near the vehicle. This window of waypoints is visible in Fig.2.1 in yellow while the green waypoints are those in the global plan.

A further addition to the local planner is the ability to shift the waypoints in the window. To do this we use the normal vector of the Frenet frame associated to the waypoint (see Fig.2.2). The Frenet frame is a frame associated to a waypoint on a trajectory, and we used the right vector which is the vector perpendicular to the trajectory. Shifting the waypoints along this vector we are able to avoid collisions when needed. The behavioral planner is responsible for setting the shift when needed

2.2.2 Overtake

One of the most important features of the new system is the ability to overtake. The system is able to detect the presence of one or multiple obstacle (in the following a cluster of obstacles) in the lane and then overtake it. Obstacles to overtake could be either vehicles or static obstacles. Contrary cars encroaching on the lane are also handled as an overtake of a dynamic obstacle to

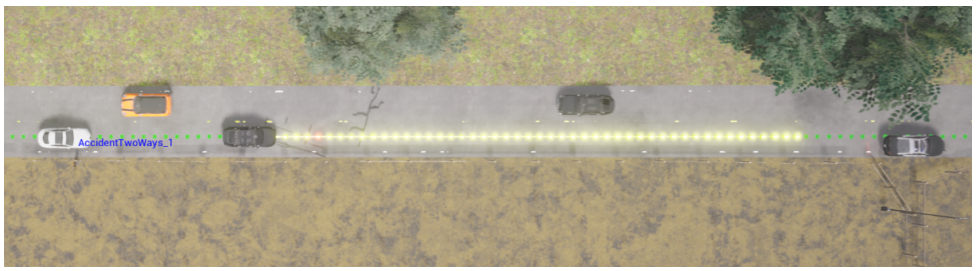


Figure 2.1: Local planner Waypoint window

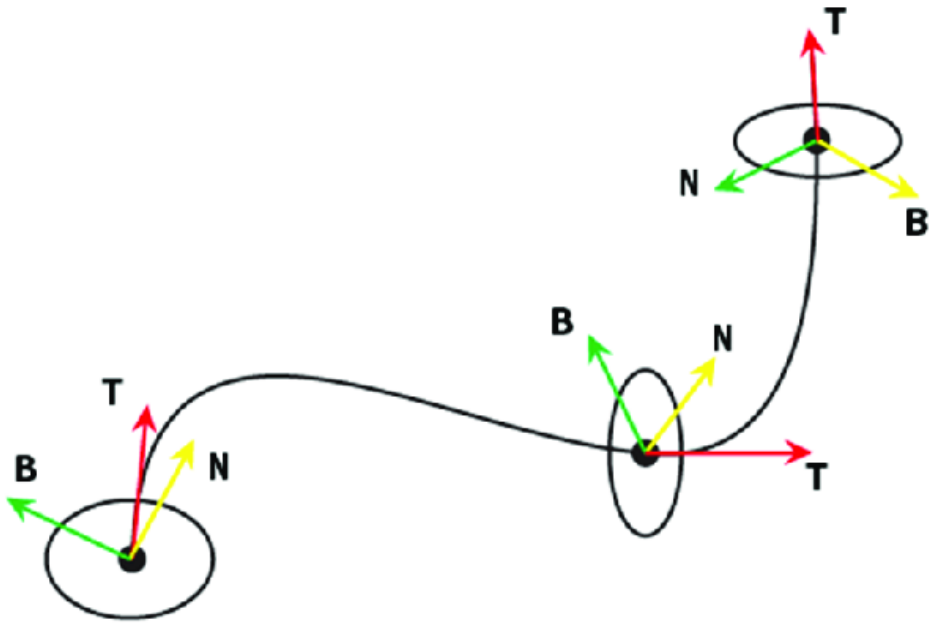


Figure 2.2: Frenet frame

the left of the vehicle.
To do this the behavioral planner must follow the following steps:

1. Detect and propose an obstacle to overtake;
2. Calculate the cluster of obstacles to be overtaken;
3. Decide whether to overtake or not based on a kinematic model-based estimation of the overtake feasibility;

If the behavioral planner decides to overtake, it delegates the local planner to calculate the path to overtake. More precisely, when the behavior planner enters the overtaking state *AvoidanceByShifting*, the local planner shifts the local waypoints of a quantity determined by the behavior planner. During the maneuver, the system uses perception modules to detect and monitor nearby obstacles. The system constantly checks if the overtaking maneuver is complete by comparing the current position of the vehicle and the obstacle position in a fixed Frenet frame. After the overtaking decision is made, the vehicle continues to overtake the obstacle until it is fully ahead of it. Once the overtaking maneuver is complete, the behavioral planner transitions to a *ReenterLane* state in witch the vehicle returns to the proper lane. Finally, the normal driving state is re-enabled. The system is also able to detect when the obstacle is no longer present and return to the original lane, effectively aborting the overtaking maneuver prematurely.
In the following sections we will dive in the details of obstacle clustering, overtaking of cyclists and overtaking of road obstacles.

2.2.2.1 Obstacle Clustering

The *ClusterDetectionManager* module orchestrates the obstacle clustering process, a critical function within autonomous driving systems aimed at obstacle detection and spatial organization

within the vehicle's operational domain. Let's explore the intricate implementations without delving into the code details.
At instantiation, the *ClusterDetectionManager* is initialized with parameters crucial for the clusterization, including the vehicle interface and clusterization margin. It leverages both the *ObstacleDetectionManager* and a *RoadSignsDetectionManager* to thoroughly examine the environment for dynamic and static obstacles, respectively. First, the process begins with collecting obstacles within a predefined detection radius starting from the cluster's reference waypoint, which can be seen as a fixed Frenet frame w.r.t. the clusterization is handled. The detection radius can be dynamically adjusted based on driving conditions and vehicle specifications. Once potential obstacles are identified, a screening process is initiated to determine their suitability for clustering. This screening considers various factors, including the alignment of obstacles with the vehicle's current lane and road, as well as their dynamic or static nature. Then, each obstacles' position is transformed into the fixed Frenet frame to facilitate spatial analysis and clustering. Once these preliminary actions are established, obstacles are sorted based on their minimum distance from the cluster reference. Sorting the obstacles is crucial for determining the order of incorporation of obstacles into the cluster. Then each obstacle behind the vehicle is discarded, as they are not relevant for the overtaking maneuver. The algorithm finally iterates through the sorted list, progressively incorporating adjacent obstacles whose distances fall within a specified clustering margin. An example cluster is show in Fig. 2.3
ClusterDetectionManager module's key role is also closely linked to the fact that without proper detection and aggregation of obstacles along the trajectory, the vehicle could be exposed to significant risks during the overtaking phase. The importance of clustering becomes evident when considering the need to accurately and timely assess the available space for overtaking. This information is crucial to determine if there is sufficient space to safely overtake, taking into account the relative speed between the autonomous vehicle and surrounding obstacles.
Another strength of the lies in its flexibility achieved through methods like *set_lane_offset*, *set_cluster_margin*, and *set_detection_distance*, enabling dynamic adjustments of clustering parameters to adapt to variable driving scenarios.

2.2.2.2 Curvilinear Abscissa

Another noteworthy feature, as it constitutes a critical point in calculating overtaking feasibility, is the computation of the obstacles' curvilinear abscissa relative to the vehicle, which is performed using the two utility functions `get_s` and `get_s_bb`. These functions ascertain the longitudinal position of obstacles along the vehicle's trajectory, which is critical for planning and executing overtakes. The first function `get_s` is responsible for calculating the longitudinal distance of a given position relative to the vehicle's current waypoint, along the direction the vehicle is oriented. This calculation involves determining the vehicle's direction vector at the given waypoint and projecting the destination's position vector onto this direction vector, resulting in a scalar value representing the distance along the direction of travel. The subsequent function `get_s_bb` extends this concept to determine the longitudinal extent of an obstacle's bounding box relative to the vehicle's current waypoint. This operation is essential for accurately assessing the size and position of obstacles in the vehicle's path, which is crucial

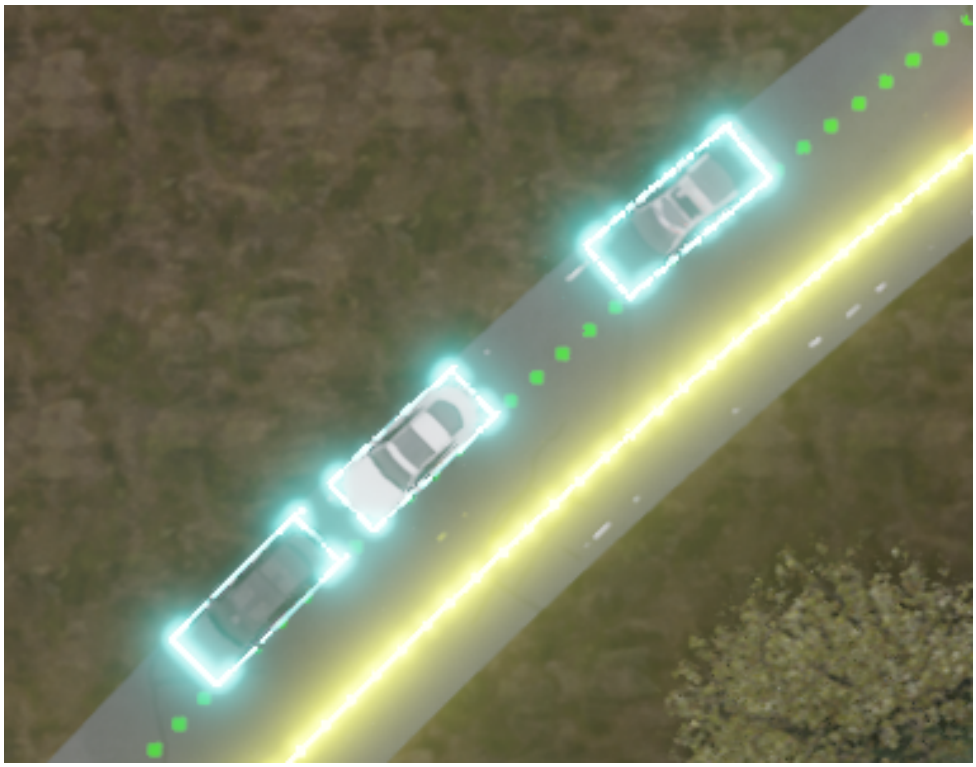


Figure 2.3: Obstacle cluster containing three stopped cars

for making informed overtaking decisions. The process is as follows:

- Direction Vector Calculation: The direction vector at the current waypoint, representing the vehicle's direction of travel, is determined.
- Position Vector Calculation: The position vector from the vehicle's current location to the obstacle's position is calculated.
- Projection: The position vector is projected onto the direction vector using the dot product, obtaining the longitudinal distance of the obstacle's position relative to the waypoint.
- Bounding Box Coordinate Determination: The real-world coordinates of the obstacle's bounding box vertices are obtained.
- Longitudinal Distance Calculation: The longitudinal distance relative to the waypoint is calculated for each vertex of the bounding box.
- Longitudinal Extent: The minimum and maximum distances among these vertices are determined, providing the longitudinal extent of the obstacle's bounding box.

The overmentioned methods are indispensable for evaluating the available space for overtaking in order to guarantee maneuver's safety.

2.2.2.3 Overtaking of cyclists

One of the overtaking cases we handled is that of cyclists. As already analyzed in chapter 1.2, cyclists are on one side of the ego vehicle lane, and based on the space they occupy, they might allow our agent to overtake them without invading the opposite lane. For this reason, the following logic was implemented: first, we check that there is sufficient space between the cyclist to be overtaken and the cars in the opposite lane. If this condition is met, the overtaking is carried out by moving the agent relative to the center of the roadway, also taking into account a safety margin. Otherwise, vehicle adapts to the cyclist's speed. To end the overtaking condition, it is verified that the cyclist has been overtaken. In case the number of cyclists to be overtaken is more than one, a cluster is created with all the cyclists to be overtaken, and the agent re-enters his path only when the last cyclist in the cluster is overtaken. An example cluster is show in Fig. 2.4. The maneuver is shown in Figs. 2.5 and 2.6.

2.2.2.4 Overtaking of road obstacles

Another overtaking scenario we handled involved obstacles occupying the agent's lane. With the word obstacle here we both refer to stopped vehciles as well as roadsigns. To overtake such obstacles, the ego vehicle is required to occupy the opposite lane.
Initially, the Frenet Optimal Trajectory planner was chosen to address this scenario [1]. This planner considers all objects occupying the agent's lane as well as vehicles moving in the opposite direction in the lane to be invaded. Using this planner, a new feasible trajectory is planned that invades the opposite lane, avoiding both vehicles in that lane and stationary obstacles in our lane, and finally returns to the initial lane. When the planner fails, for example due to vehicles coming from the lane to be invaded, the vehicle brakes and the agent waits in its lane until overtaking becomes feasible. However, the planner was ultimately abandoned because it repeatedly failed as it lacked the recursive feasibility property that is fundamental to guarantee when using this kind of optimizers with a receding horizon. In other words, there was no guarantee that the planner would find a solution in the next instant after finding one in the previous instant. At a certain point during the maneuver, the planner failed in optimization, leading to the need to address issues that rendered its use impractical. Additionally, the planner could only consider rectangular obstacles aligned with the Cartesian axes, which was a limitation because diagonal obstacles would occupy more space than their actual size.

As a consequence, overtaking was eventually treated like other overtaking scenarios, namely by performing a shift. To enable the treatment of this overtaking with a shift, a complex logic was introduced to check the feasibility of the overtaking maneuver. This logic will be described in the next section. Figures 2.7 and 2.8 show the overtaking of static obstacles, while Figures 2.9, 2.10 and 2.11 show the overtaking of dynamic obstacles.

2.2.2.5 Evaluating overtake feasibility

The aim is to assess the feasibility of overtaking safely and efficiently. This feature is implemented through the *ClusterDetectionManager* module and its associated functions for overtaking distance calculation and feasibility evaluation. At the core of this feature lies the

Figure 2.4: Cluster containing two cyclists to be overtaken

Figure 2.5: Overtaking of two cyclists

Figure 2.6: Reentering the lane after overtaking the cyclists

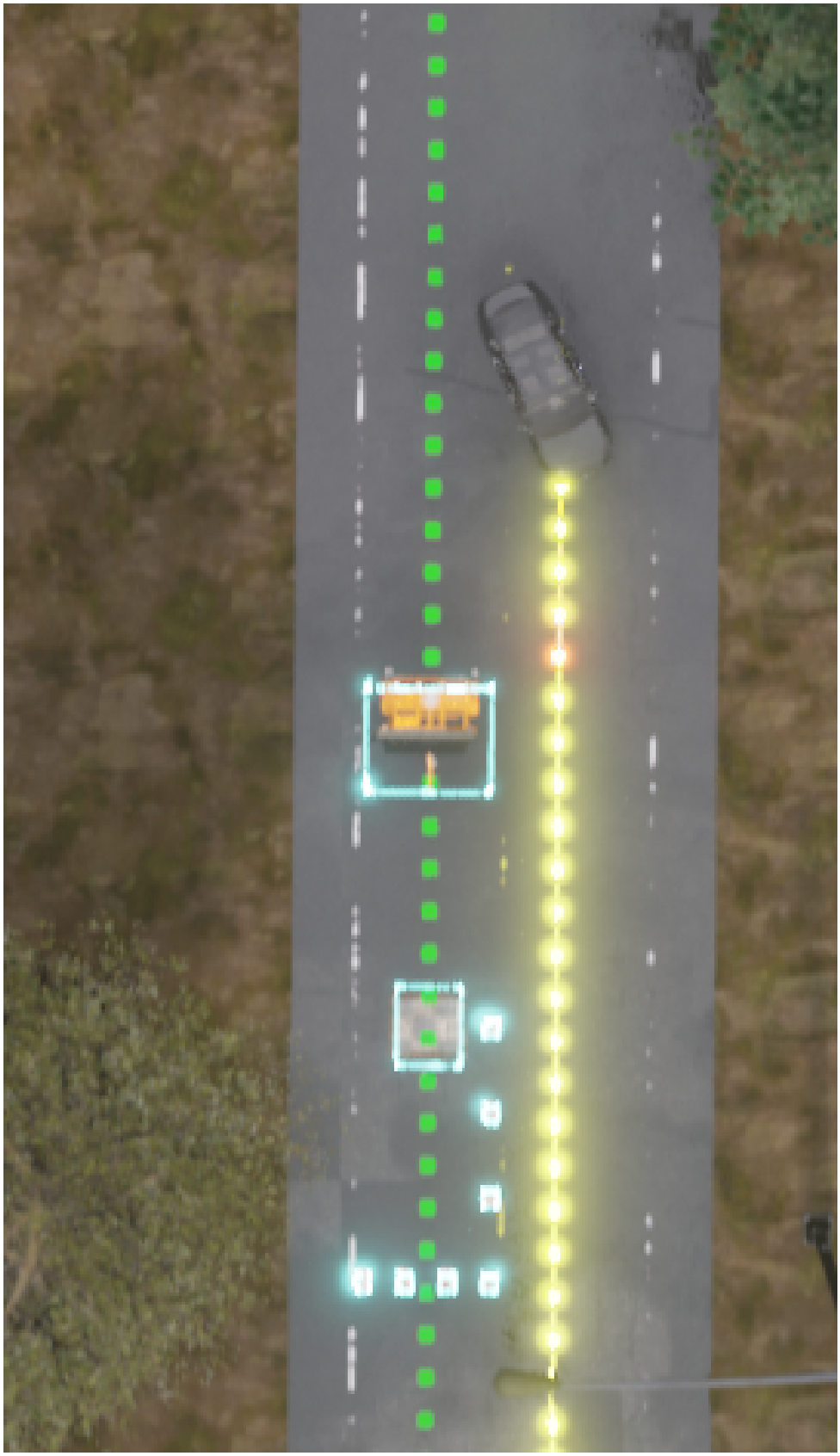


Figure 2.7: Static obstacles to be overtaken

Figure 2.8: Overtakeing of static obstacles

Figure 2.9: Dynamic obstacles to be overtaken

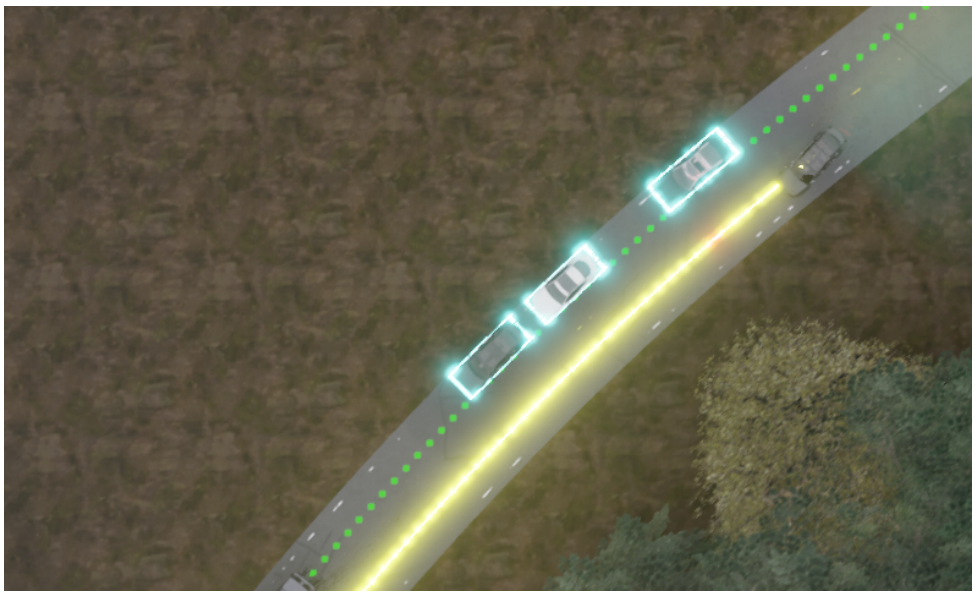


Figure 2.10: Overtakeing of dynamic obstacles - 1

Figure 2.11: Overtakeing of dynamic obstacles - 2

clustering approach described in section 2.2.2.1, which aggregates obstacles based on their spatial proximity to the autonomous vehicle. Once the overtaking distance is determined in the fixed Frenet frame, the function for overtaking distance calculation determines the margin of available space to complete the maneuver safely. This calculation is crucial for accurately and timely assessing the feasibility of overtaking. The function for overtaking feasibility evaluation examines the road situation and the presence of vehicles in the opposite lane. If no vehicles are detected in the opposite lane and the overtaking space is sufficient, the feature deems overtaking feasible and returns a positive outcome. Conversely, if vehicles are detected in the opposite lane or the overtaking space is inadequate, the system evaluates the situation to determine if it is safe to proceed with the maneuver. This involves predicting the time that the agent will take to overtake the obstacle and return to the original lane, as well as the relative speed between the autonomous vehicle and the surrounding obstacles. Then the future position of the incoming vehicles is estimated to ensure that the overtaking maneuver can be completed without causing a collision. This comprehensive evaluation process is essential for ensuring the safety and efficiency of overtaking maneuvers, minimizing the risk of accidents and promoting a smoother driving experience.

2.2.3 Vehicles invading the agent's lane

An additional scenario to handle is when vehicles traveling in the opposite lane partially encroach on the ego vehicle's lane, due to road works or otherwise. In this case, these vehicles move relative to the center of their travel lane toward our agent. To handle this scenario, first, it is checked that vehicles coming from the opposite lane have actually moved relative to the center of their lane. If this condition is met, a shift of the agent is made so as to maintain a sufficient safe distance from the vehicles in the opposite lane, taking into account the fact that the agent should not go off the road. In the event that the officer would go off the road, for safety reasons, the agent doesn't perform the maneuver. For these feature, both the obstacle clustering and the local planner's shifting capabilities described in section 2.2.2.5 are exploited. Figures 2.12, 2.13, 2.14 and 2.15 show the overtaking of vehicles invading the agent's lane.

2.2.4 Re-entering the lane

This behavior planner state allows the agent to rejoin its lane post a deviation maneuver. This capability is crucial to ensure the vehicle's safe return to regular lane driving following obstacle evasion or overtaking maneuvers. Without this capability, the vehicle risks lingering in precarious positions on the road, increasing the likelihood of collisions. Upon entering the *ReenterLane* state, the system continually monitors the vehicle's position relative to the global navigation plan. Leveraging data from perception modules and the local planner, the system validates whether the vehicle has re-entered the same road and lane as the planned waypoint. The process of confirming lane reentry involves comparing the vehicle's road and lane identifiers with those of the waypoint. If they correspond, the lane reentry is deemed complete, enabling the vehicle to resume its regular driving trajectory. This maneuver is show in Figures 2.16 and 2.17.

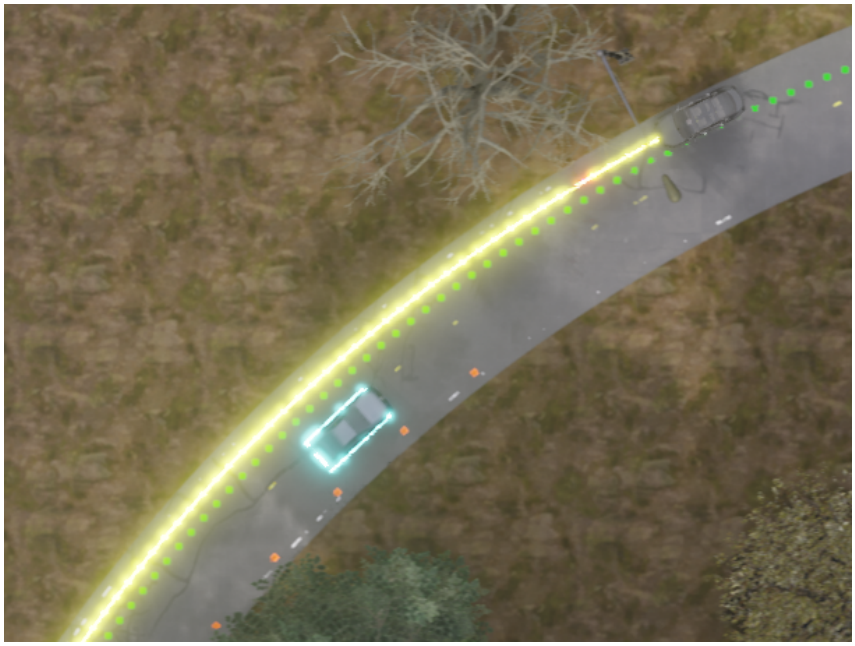


Figure 2.12: Overtaking of vehicles invading the lane 1

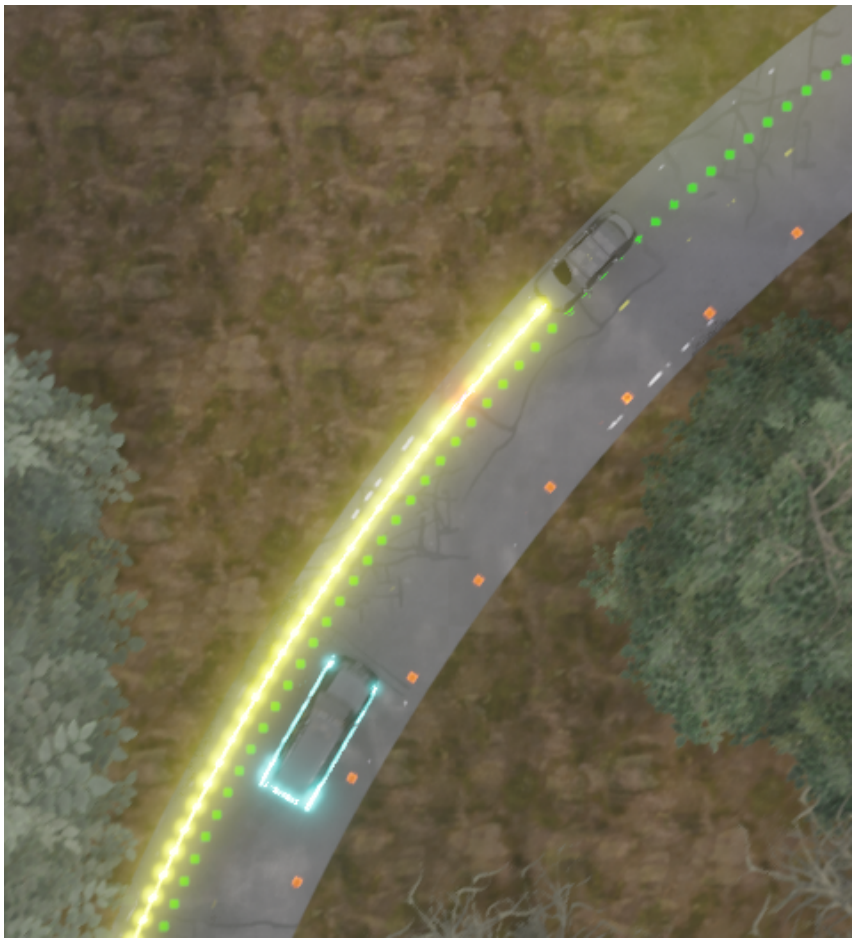


Figure 2.13: Overtaking of vehicles invading the lane 2

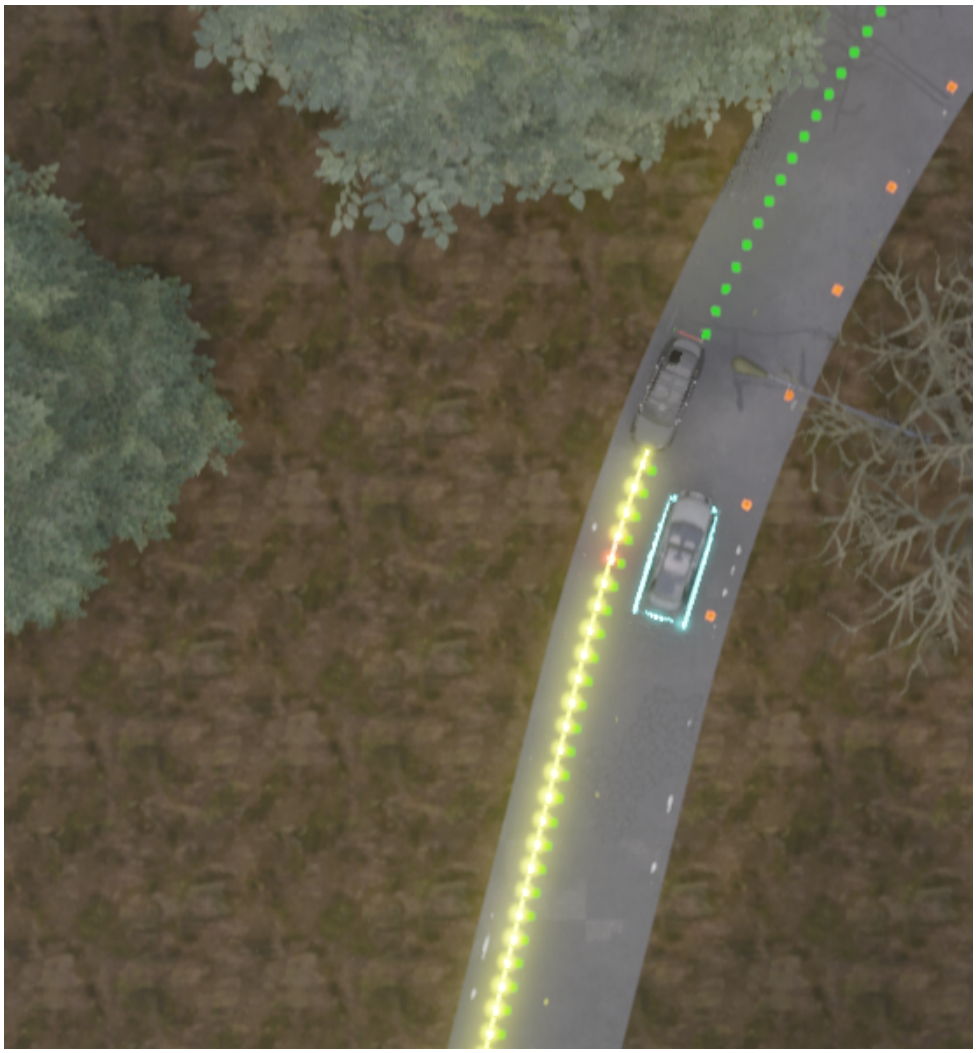


Figure 2.14: Overtaking of vehicles invading the lane 3

Figure 2.15: Overtaking of vehicles invading the lane 4

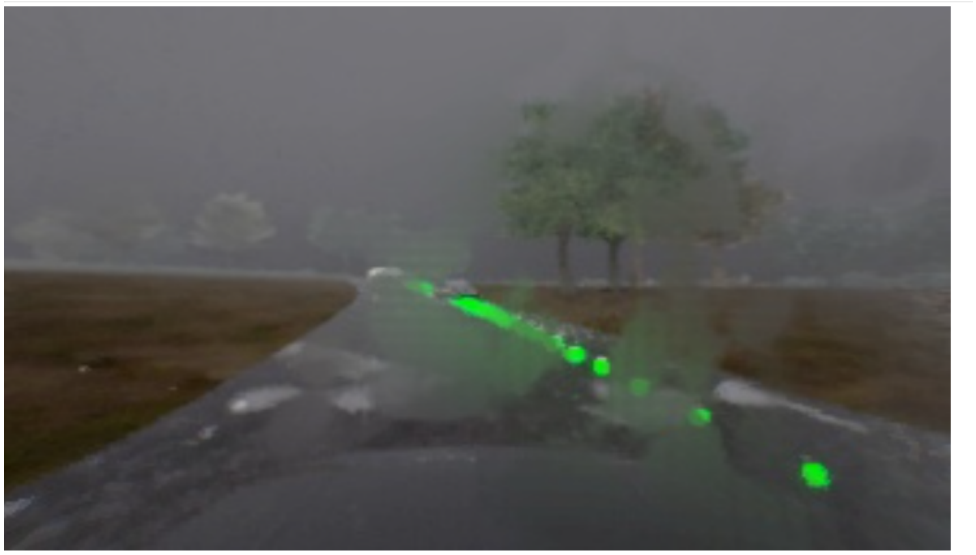


Figure 2.16: Re-entering the lane 1

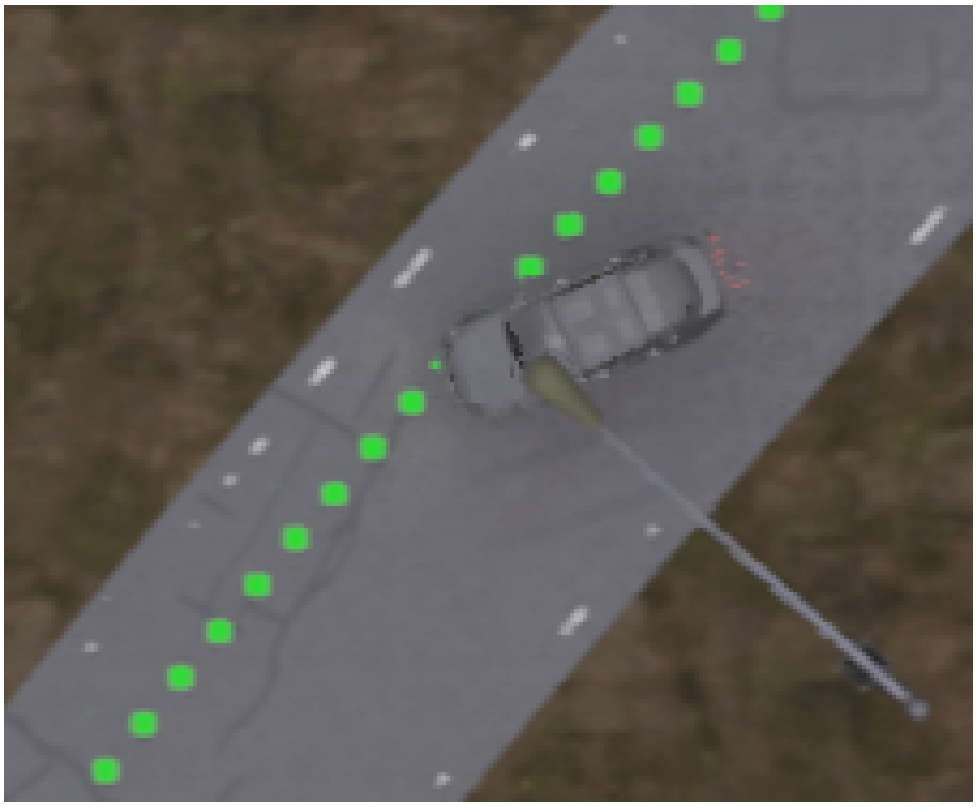


Figure 2.17: Re-entering the lane 2

2.2.5 Decelerate then stop

The stop sign scenario is a crucial situation for our autonomous vehicle to manage. A behavioral planner state was implemented in which the vehicle must detect a stop sign and respond by smoothly decelerating to come to a complete stop before the intersection. Successfully handling this scenario is vital for ensuring road safety, complying with traffic laws, and preventing potential collisions. To manage this driving situation it has been implemented the *DecelerateThenStop* feature which allows the autonomous vehicle to detect stop signs and perform a gradual deceleration followed by a complete stop near such signs. Without this capability, the vehicle would risk failing to comply with traffic regulations, endangering road safety and increasing the risk of accidents. The vehicle's perception system identifies the position of a stop sign and communicates this information to the behavioral planning module. The feature uses the uniformly accelerated motion equation to calculate the required deceleration:

$$a = -\frac{{v_0}^2}{2 \cdot d} \tag{2.1}$$

where v_0 is the initial speed of the vehicle and `d` is the remaining distance to the stop sign. This equation is derived from the uniformly accelerated motion law:

$${v_f}^2 = {v_0}^2 + 2a(d - d_0) \tag{2.2}$$

with $v_f = 0$ at the position of the stop sign. Using the calculated deceleration, the vehicle's target speed is continuously updated. The target speed at a given moment is determined by the following equation:

$$v(t) = v_0 + a \cdot (t - t_0) \tag{2.3}$$

where `t` is the current time and t_0 is the initial time. The distance between the vehicle and the stop sign is constantly monitored to ensure that the deceleration is adequate and that the vehicle stops exactly at the stop sign's position. The feature ensures that the vehicle stops safely at stop signs, reducing the risk of collisions, and improves the driving experience with a gradual deceleration, avoiding abrupt braking and enhancing passenger comfort.

2.2.6 Wait at Stop

After decelerating the vehicle must wait at the stop sign for a certain amount of time before proceeding. A behavioral planner state was implemented to allow the vehicle to manage the scenario of waiting at a stop sign. This functionality ensures that the vehicle remains stationary for a predetermined period, simulating a real-world behavior of obeying stop signs and traffic regulations. This is due to the fact that in order to avoid the penalty, when an autonomous vehicle approaches a stop sign, it must not only come to a complete stop but also wait for a sufficient amount of time before proceeding in order to check if it can occupy the junction. When the vehicle transitions into the *WaitAtStop* state, the system keeps track of the flowing of time. The vehicle remains in this state until the predefined wait time has elapsed. During this period, the vehicle's throttle is set to zero, and maximum brake is applied to ensure it stays stationary. Once the wait time has passed, the system updates an internal flag indicating that the waiting

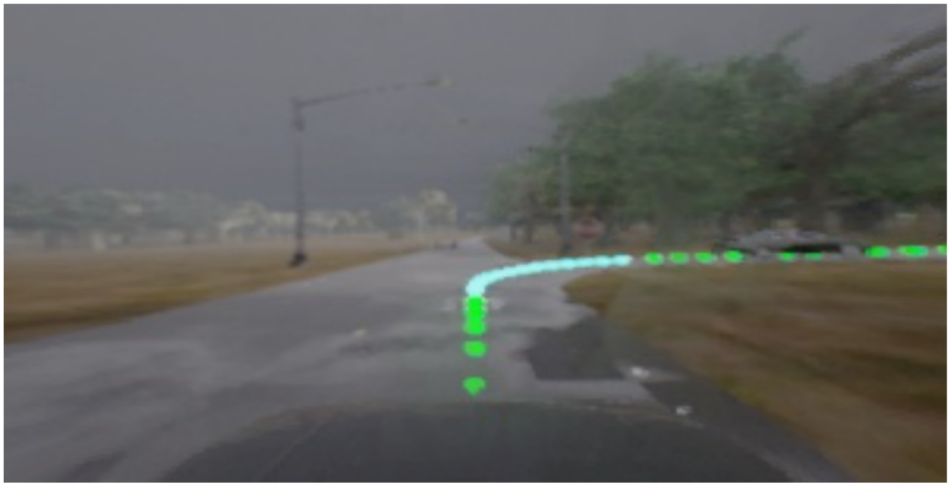


Figure 2.18: Decelerate then stop

period is over, and the vehicle can proceed to the next state. This transition is managed by the behavioral planner, ensuring the vehicle resumes motion in a controlled and safe manner. Figure 2.19 shows the vehicle waiting at a stop sign.

2.2.7 Junctions

A state in the behavioral planner was implemented to manage the crossing of intersections. This capability is essential to ensure the vehicle can navigate through intersections safely and efficiently, avoiding collisions and adhering to traffic rules. In a driving context, crossing an intersection is a complex situation that requires the vehicle to detect and predict other vehicles'trajectories in the intersection area and respond accordingly. Without this capability, the vehicle risks crossing the intersection dangerously, increasing the risk of accidents. However, we considered a simpler solution. When the vehicle enters the *Junction* state, the system sets a target speed and identifies the intersection area. During this state, the system uses data provided by the vehicle detection manager to obtain a list of vehicles within a certain distance from the intersection. If any vehicle is actually within the intersection area and is moving at a speed above a minimum threshold, the system performs an emergency stop to avoid collisions. This involves the local planner intervening to apply the appropriate control commands to stop the vehicle safely. If no hazardous vehicles are detected in the intersection area, the system sets the vehicle's speed to the previously determined target speed and allows the vehicle to cross the intersection safely and in a controlled manner. This rather conservative logic has been chosen to manage intersections to ensure that the vehicle crosses them under the safest possible conditions. Figure 2.20 shows the vehicle crossing an intersection, while Figure 2.21 shows the bounding box of the intersection.

Figure 2.19: Wait at Stop

Figure 2.20: Agent at a junction

Figure 2.21: Junction's bounding box

2.2.8 Follow Leading Vehicle

The performances of the *FollowLeadingVehicle* feature were considered to be insufficient so we decided to improve it. This functionality has been enhanced by integrating an Adaptive Cruise Control (ACC) called Intelligent Driver Model (IDM) as described in [2]. The baseline functionality allowed the vehicle to maintain a constant and steady speed using longitudinal control and to approximately follow the lead vehicle with a discontinuous function of the distance. In this approach, the target speed was set by means of a continuous-time car following model. With the addition of Adaptive Cruise Control (ACC), the system now dynamically adjusts the vehicle's speed to maintain a safe distance from other vehicles while driving. Instead of a static target speed, the ACC uses a predictive model of surrounding vehicle dynamics to calculate an adaptive target speed in real-time. This model takes into account factors such as the distance and relative speed of the leading vehicle, as well as the target speed. Thanks to this improvement, the vehicle can now continuously vary its speed to dynamically adapt to the leading vehicle. Some examples of the ACC in action are shown in Figures 2.22 and 2.23.

Figure 2.22: Follow Leading Vehicle, front view

Figure 2.23: Follow Leading Vehicle, top view

2.3 Resulting Finite State Machine

At the end of this chapter we have a finite state machine that is composed of the following states, that summarize the agent's behavior:

- **Lane following**: the agent follows the lane and set the target speed to the speed limit;
- **Overtake with obstacles**: the agent overtakes road objects, cyclists and stationary vehicles, while avoiding collisions;
- **Re-enter lane**: the agent re-enters the lane after overtaking;
- **Avoidance by shifting**: the agent shifts to the left or right to avoid a collision;
- **Decelerate**: the agent decelerates when approaching a stop sign, and then stops;
- **Wait at stop**: the agent waits at the stop sign for a few seconds;
- **Junction**: the agent manages the junction waiting for the right moment to cross;
- **Following leading vehicle**: the agent follows the leading vehicle at a safe distance;
- **Emergency stop**: the agent stops immediately when an obstacle is detected in front of it, or when the leading vehicle stops.

We can see the finite state machine in figure 2.24.

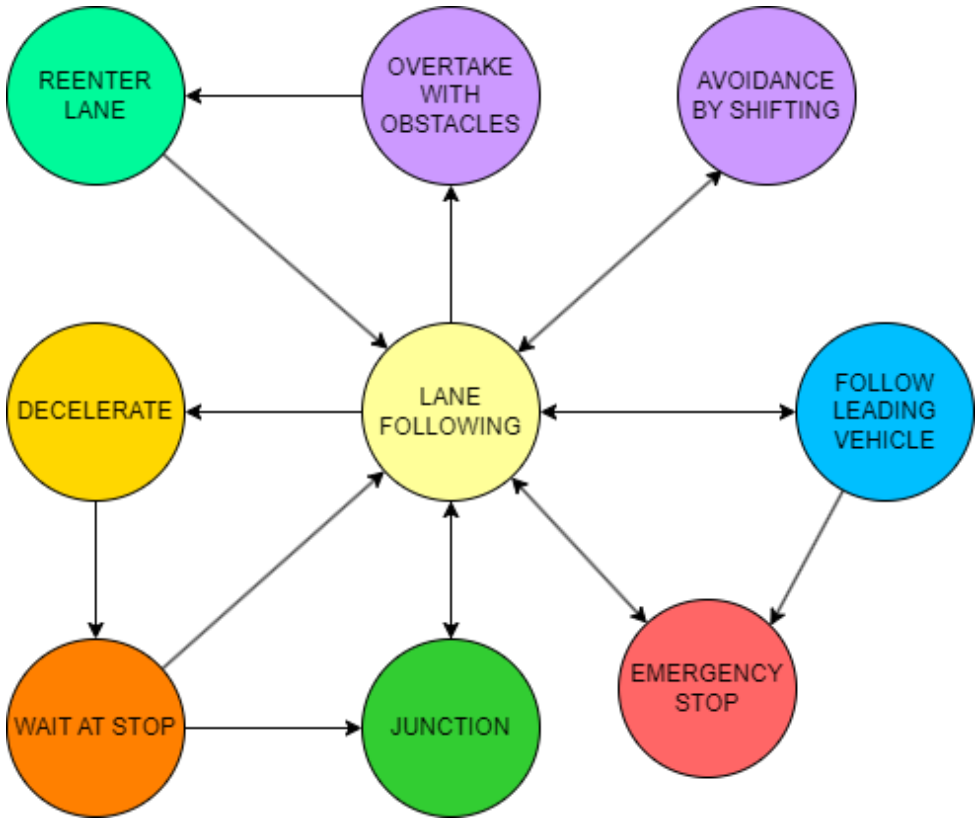


Figure 2.24: Finite State Machine

CHAPTER 3

ANALYSIS OF THE IMPROVED AGENT

3.1 Scenario Analysis

3.1.1 AccidentTwoWays

As seen in Ch. 1.2.1, in this scenario, an accident involving two vehicles and a police car completely occupies the agent's lane for a long stretch of road. The baseline agent goes to rear-end the police car, and as a result, he gets stuck and did not continue until the scenario timeout expired. To overcome this major problem, the overtaking feature was introduced, in which case it can be seen how crucial the use of the cluster is (Ch. 2.2.2). As the results show, this scenario was also overtaken. The results are shown in the Figs. 3.1, 3.2 and 3.3.

3.1.2 BlockedIntersection

In this type of scenario, the agent finds a blocked intersection. The baseline agent not handle intersections at all. It is also observed that in scenario 19 (Ch. 1.2.1), the agent encounters a stop sign, which is not respected. To solve these issues, several functions were implemented. First, as the stop is approached, the agent switches to the *DecelerateThenStop* state (Ch. 2.2.5),

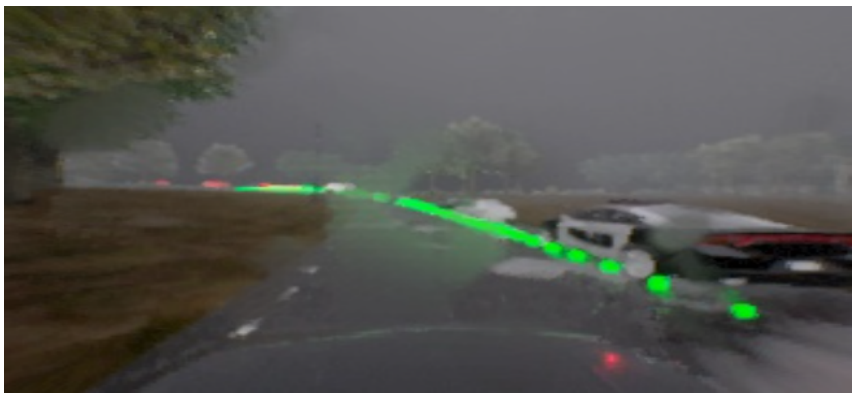


Figure 3.1: Scenario AccidentTwoWays resolution - camera

Figure 3.2: Scenario AccidentTwoWays resolution - map

Figure 3.3: Scenario AccidentTwoWays resolution - map2

Figure 3.4: Scenario BlockedIntersection resolution - camera

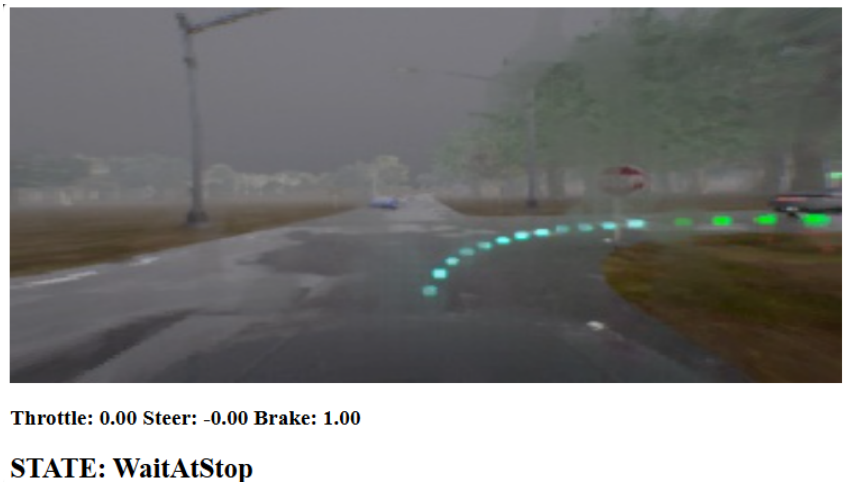


Figure 3.5: Scenario BlockedIntersection resolution - camera2

in the case where there is a stop signal at the intersection, to gradually slow down. Once stopped at the stop sign, the agent switches to the *WaitAtStop* state (Ch. 2.2.6), where it waits 3 seconds before restarting. Finally, to properly navigate the intersection, the agent transitions into the *Junction* state (Ch. 2.2.7), which ensures that the intersection is not passed until it is clear. As the results show, this scenario was also passed, and the agent correctly handled the intersection. The results are shown in the Figs. 3.4, 3.5, 3.6 and 3.7.

3.1.3 ConstructionObstacleTwoWays

As described above (Ch. 1.2.1), in this type of scenario, the roadwork sign completely occupies the agent's lane. The baseline agent runs into this obstacle and then does not continue until the timeout expires. This problem was solved by going to implement obstacle overtaking, as described in Ch. 2.2.2.4. So our agent is able to recognize that the road is obstructed and safely overtakes. We can see the results in the Figs. 3.8 and 3.9.

Figure 3.6: Scenario BlockedIntersection resolution - camera3

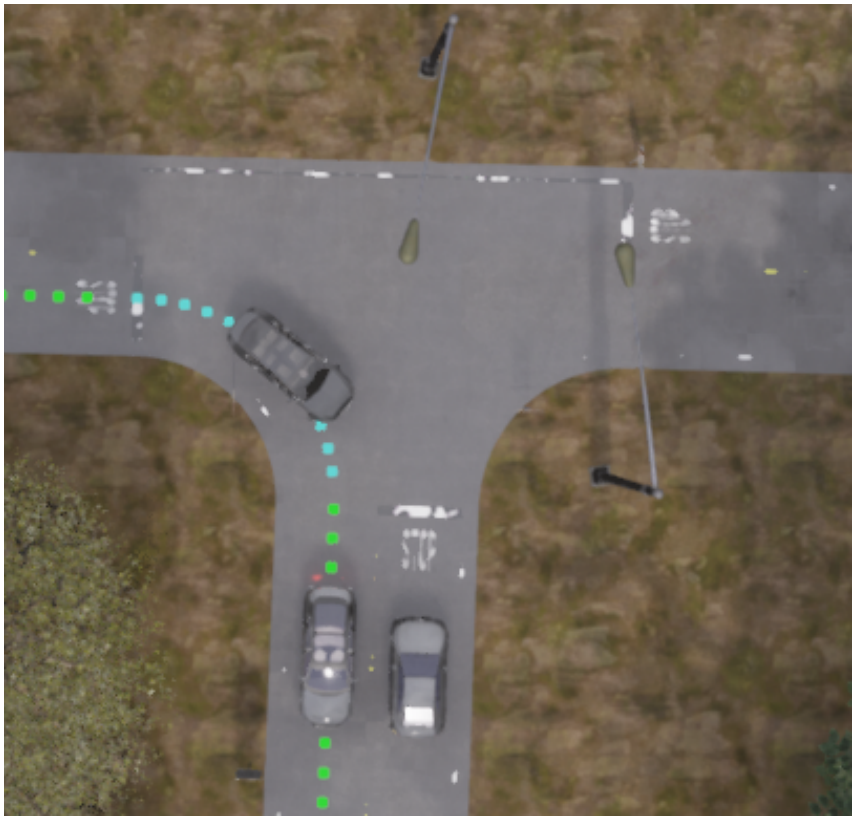


Figure 3.7: Scenario BlockedIntersection resolution - map

Figure 3.8: Scenario ConstructionObstacleTwoWays resolution - camera

Figure 3.9: Scenario ConstructionObstacleTwoWays resolution - map

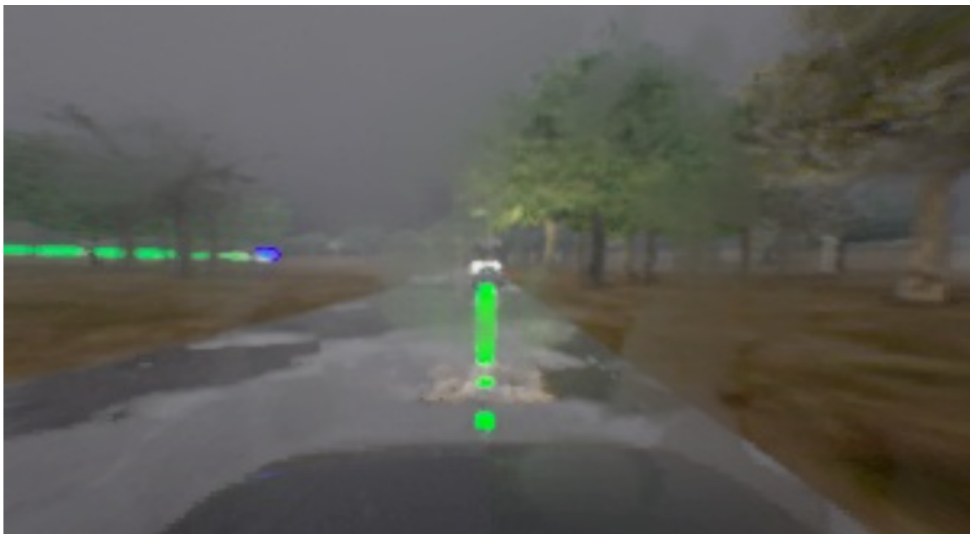


Figure 3.10: Scenario ControlLoss resolution - camera

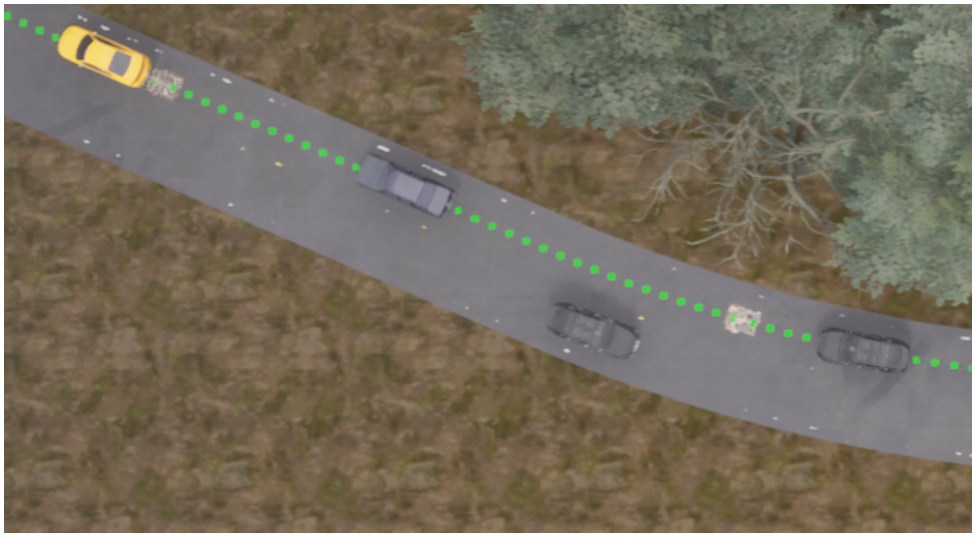


Figure 3.11: Scenario ControlLoss resolution - map

3.1.4 ControlLoss

This particular scenario did not present problems in the baseline agent, as described in Ch. 1.2.1, and, even after adding all the features, continues to cause no problems or penalties. However, by fine-tuning the controllers, we made the car more stable. The results are shown in the Figs. 3.10 and 3.11.

3.1.5 DynamicObjectCrossing

In this type of scenario, very often the baseline agent behaves well, so it stops as the pedestrian passes, but in some cases, especially in scenario 5 (Ch. 1.2.1), the agent runs over the pedestrian, most likely due to a scenario bug, so the pedestrian sweeps dead, thus resulting in the `collision_pedestrian` penalty. The refactoring done along with other features introduced, coupled with better handling of state transitions, allowed us to resolve this scenario even in previously problematic cases, causing our agent to enter the *EmergencyStop* state correctly. As

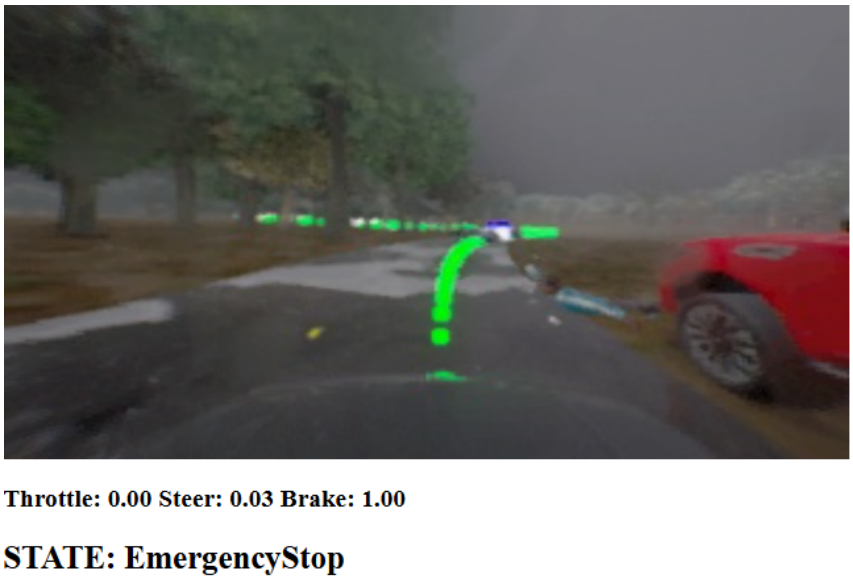


Figure 3.12: Scenario DynamicObjectCrossing resolution - camera

can be seen from the results, this scenario was passed. The results are shown in the Figs. 3.12 and 3.13.

3.1.6 HardBreakRoute

In some cases of this type of scenario in the baseline agent, collisions were occurring after intersections. The refactoring done along with other features introduced, along with better handling of state transitions and intersections, allowed us to solve this scenario even in previously problematic cases by having the agent enter the *Junction state*. As can be seen from the results, this scenario was overcome.

3.1.7 HazardAtSideLaneTwoWays

As described in the baseline analysis chapter (Ch. 1.2.1), in this type of scenario there are bicyclists occupying part of the agent's lane, thus slowing the agent down. In this scenario, the baseline agent slows down a lot, sometimes gets stuck behind the cyclists, goes into the emergency braking state, and then resumes driving. Also, when the cyclists stop, in some cases the agent overtakes them and comes very close and sometimes hits them, while in other cases he gets stuck behind them. This is always solved with the overtaking functionality, and in particular with a state of the finite-state automaton created specifically to handle cyclists (Ch. 2.2.2.3). In fact, the agent is now able to correctly overtake all such scenarios. The results are shown in the Figs. 3.14 and 3.15.

3.1.8 InvadingTurn

Although there were no particular penalties in the baseline agent, apart from the one involving the minimum speed of the ego vehicle as seen in Ch. 1.2.1, it was decided to make improvements to this type of scenario, because it was evaluated as dangerous behavior and with the newly introduced features did not require too much effort. In fact, to solve such scenarios,

Figure 3.13: Scenario DynamicObjectCrossing resolution - map

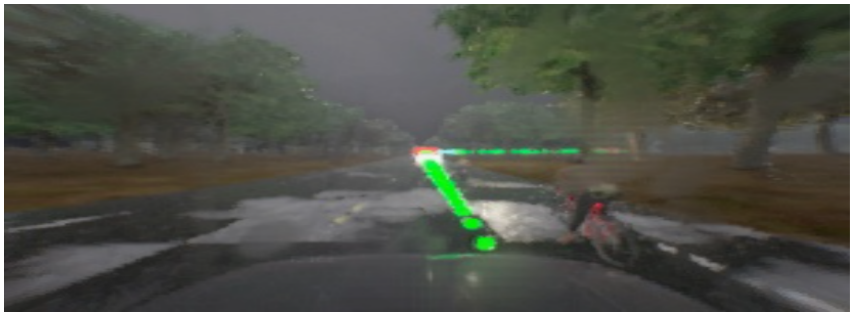


Figure 3.14: Scenario HazardAtSideLaneTwoWays resolution - camera

Figure 3.15: Scenario HazardAtSideLaneTwoWays resolution - map

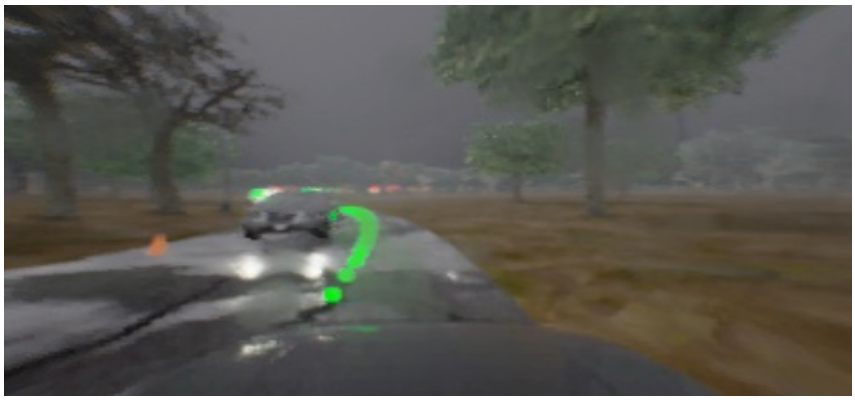


Figure 3.16: Scenario InvadingTurn resolution - camera

we go for a shift, taking advantage of the new functionality explained in Ch. 2.2.3. If the vehicle in the lane opposite to the agent's lane comes dangerously close to our lane, the agent performs a displacement to stay in his own lane but at a safe distance from the vehicles in the opposite lane. As a result, once an adequate distance is maintained from the vehicles in the opposite lane, the agent can safely maintain the set speed and thus no longer get the minimum speed penalty. This result also emerges from the experimental results and is shown in the Figs. 3.16 and 3.17.

3.1.9 NonSignalizedJunctionRightTurn

This type of scenario was solved in the same way seen for BlockedIntersection (Ch. 3.1.2). In fact, this was a scenario in which we had problems with intersections and stop signs. This result also emerges from the experimental results. The results are shown in the Fig. 3.18.

3.1.10 ParkedObstacleTwoWays

For this scenario, in which the baseline agent rear-ended the stationary vehicle in our lane, as seen in Ch. 1.2.1, it was solved using the logic of overtaking and grouping. This is also evident from the experimental results and is shown in the Fig. 3.19.

3.1.11 VehicleTurningRoute

This type of scenario involves a cyclist crossing the road. The baseline agent does not recognize the cyclist perfectly, so several times it runs over the cyclist. So a recognition of cyclists crossing the road was implemented to solve this problem. From the experimental results, the correct behavior of the agent is observed in the Figs. 3.20, 3.21 and 3.22.

3.1.12 VehicleTurningRoutePedestrian

This scenario falls fully within the scope of the VehicleTurningRoute scenario (Ch. 3.1.11). In fact, it was solved in the same way. The only difference is that in this case, there is a pedestrian crossing the road instead of a cyclist. The results are shown in the Figs. 3.23 and 3.24.

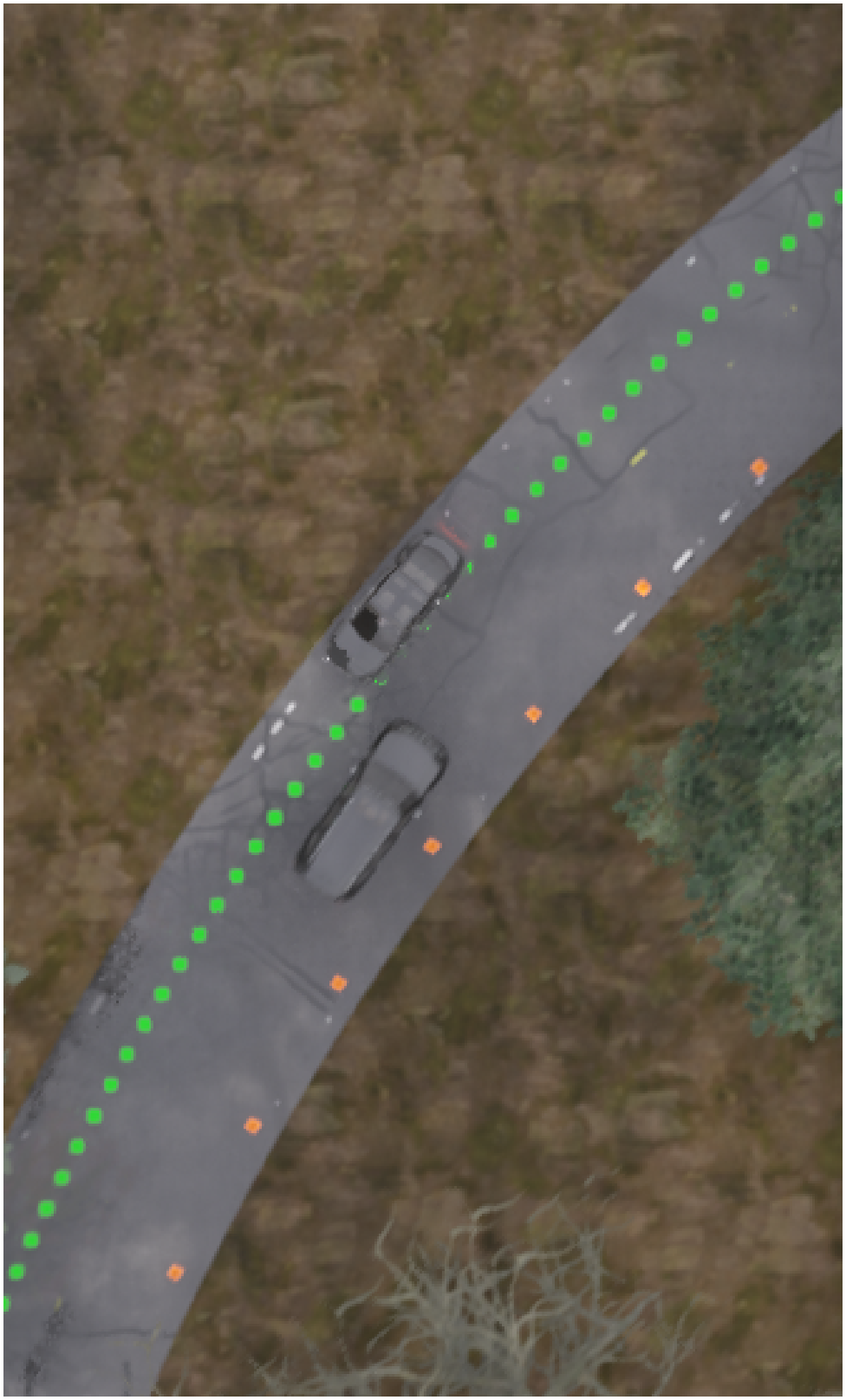


Figure 3.17: Scenario InvadingTurn resolution - map

Figure 3.18: Scenario NonSignalizedJunctionRightTurn resolution - camera

Figure 3.19: Scenario ParkedObstacleTwoWays resolution - map

Figure 3.20: Scenario VehicleTurningRoute resolution - camera

Figure 3.21: Scenario VehicleTurningRoute resolution - map

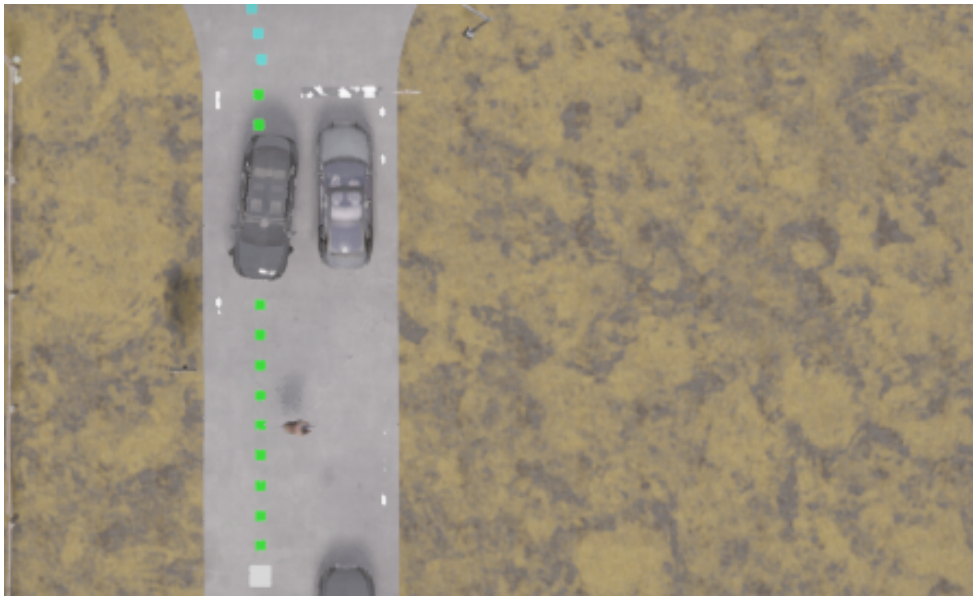


Figure 3.22: Scenario VehicleTurningRoute resolution - map2

Figure 3.23: Scenario VehicleTurningRoutePedestrian resolution - camera

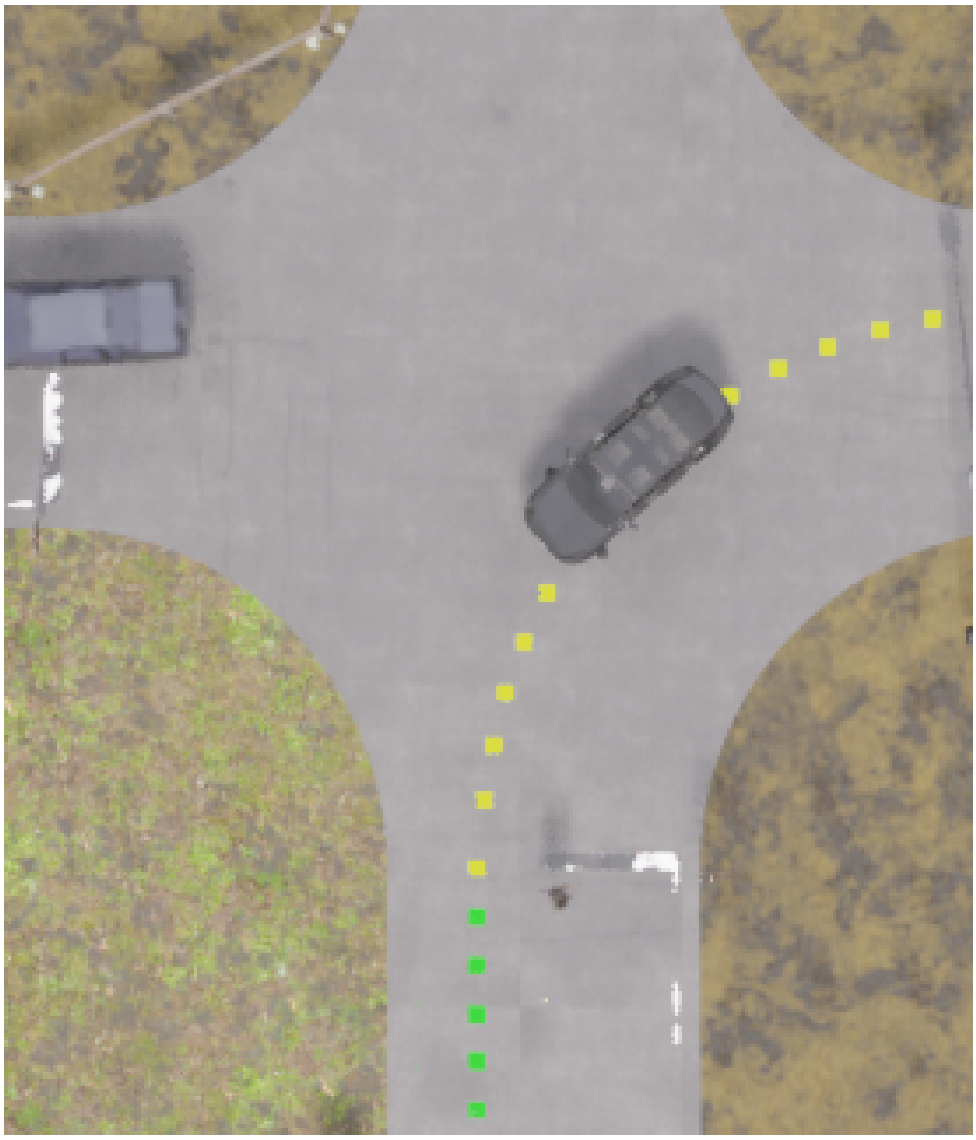


Figure 3.24: Scenario VehicleTurningRoutePedestrian resolution - map

3.2 Route Analysis

In this section we provide the final agent evaluation on the provided routes. The videos of the agent proving the claimed results are attached to this report. Note that the presented results are the best performances the agent can achieve, and they are reached in almost every simulation. However, the agent's performance may vary depending on the randomness of the simulation environment. One could argue that a simulation environment like CARLA leaderboard should be able to deterministically reproduce the same results to ensure a fair comparison between different agents. However, this is not the case for the CARLA leaderboard, and the randomness of the simulation environment is a known issue. All of the remaining problems (that manifest very rarely) are due to the randomness of the simulation environment and fall in two categories:

1. Problems that are not imputable to our agent. This is the case of the vehicle invading the lane of the agent in route 4. As we will discuss below, these random events are not imputable to our agent, and they are not a consequence of the agent's behavior, so we didn't consider the (rare) cases in which it occurred. Notice that only one problem is left in this category;

2. Problems that are imputable to our agent, but we couldn't reproduce. Simulations are slow and the very few problems that are left can only be reproduced simulating the full route multiple times. As those problems are very rare and the simulation is slow, we couldn't reproduce them, so we couldn't debug them. This is the reason why we consider the randomness of CARLA leaderboard's scenarios a fault.

In our experiments we observed that the only element of randomness should be the direction chosen by the other agents at the intersections.

3.2.1 Route 1

These are the following results obtained by executing the route 1 with the agent we implemented:

- The score route is 100.0.
- The score penalty varies between 0.89 and 0.98. [1]
- Therefore the total score varies from 89.0 and 98.0.

Considering that the only issue is due to minimum speed and that we decided not to solve this problem explicitly, we reached our goal. It should be noted that in scenario 21 (Ch. 1.2.1) there is an intersection where the agent has to traverse a junction that is sometimes highly trafficked, and therefore, for the way we have implemented our agent, the time for passing this scenario is random.

[1] When the agent is in the final intersection sometimes the traffic conditions are such that the agent has to wait for a long time before passing the intersection. The quantity of time the agent loses is a random fact that depends on traffic conditions and causes the agent to lose a variable amount of points.

3.2.2 Route 4

These are the following results obtained by executing the route 4 with the agent we implemented:

- The score route is 100.0.
- The score penalty is 1.0.[2]
- Therefore the total score is 100.0.

This means that the route is fully completed without penalties. However, we want to point out a problem concerning the penultimate intersection of this route (1.2.2). Specifically, often other vehicles invade the opposite lane going to hit the ego vehicle, when turning into the road where the ego vehicle is present, even if it is stationary. This, however, is counted as a penalty of type `collisions_vehicle`.

[2]When the agent is in second-to-last intersection rarely a car invades the lane of the agent. This is a random fact that isn't expected in the scenario and causes the agent to lose few points due to the collision that is not imputable to the agent. However, we couldn't reproduce this problem reliably and therefore we didn't consider it.

CHAPTER 4

FUTURE DEVELOPMENTS AND CONCLUSIONS

4.1 Future Developments

In this section, we will discuss the future developments that can be made to the system to improve the performance of the agent. The following are some of the future imporvements that can be made to the system:

- Formulation of an optimal control problem in the local planner for managing collision avoidance. As we said, we already tried to integrate an optimizer for the local planner but it suffered from lack of recursive feasibility;
- Utilizing of Model Predictive Control (MPC) for lateral and longitudinal control at the same time should be a consistent improvement, as this techniques allows for a more accurate prediction of the vehicle's behavior as well as considering the saturation of the actuators;
- Realization of real perception and localization modules for the agent should be considered if we would hope to deploy the agent in a real-world scenario;
- Using a behavioral tree for the behavioral planner instead of the current finite state machine should be considered but it is not necessarily an improvement. The behavioral tree is more flexible and can be used to model more complex behaviors, but it is also more complex to implement and maintain;

One of the critical aspects of this project is the behavior of the autonomous driving agent at intersections. Currently, the vehicle stops if the intersection is occupied by any vehicle, even if the vehicle does not obstruct its trajectory. This overly cautious behavior results in the vehicle halting unnecessarily when other cars are passing through the intersection without posing any risk of collision. A significant improvement would be to implement tracking for vehicles,

pedestrians, and riders. By estimating their future positions, the agent can make more informed decisions and stop only when there is an actual risk of collision. This approach would involve:

- Enhanced Perception Module: Upgrade the Perception module to include predictive tracking algorithms. These algorithms would monitor the current positions and velocities of vehicles, pedestrians, and riders. Use machine learning models or Kalman filters to predict the future trajectories of these objects.
- Trajectory Prediction: Develop a system that can estimate where each detected object will be in the next few seconds. Incorporate these predictions into the decision-making process, allowing the agent to differentiate between objects that pose a collision risk and those that do not.
- Dynamic Decision-Making: Integrate the predicted trajectories into the Planners module. The agent should dynamically assess potential collision risks based on the predicted positions of moving objects.Implement a decision-making algorithm that allows the vehicle to proceed through an intersection if the predicted paths of other vehicles do not intersect with its own path.
- Safety and Efficiency Balance: Ensure that the new system maintains a balance between safety and efficiency. The goal is to prevent unnecessary stops while still ensuring the highest level of safety. Conduct extensive testing in various intersection scenarios to fine-tune the system and verify that it improves the overall performance without compromising safety.

By incorporating these improvements, the autonomous driving agent can handle intersections more intelligently, reducing unnecessary stops and improving overall driving efficiency. This enhancement will make the agent's behavior more human-like, where decisions are based on a better understanding of the surrounding environment and potential risks

4.2 Conclusion

At the end, we can conclude that the agent we implemented is able to complete the routes provided in the baseline with an almost perfect score. Each objective we pursued was accomplished successfully.

LIST OF FIGURES

BIBLIOGRAPHY

[1] Unni Krishnan R Nair et al. "SVM Enhanced Frenet Frame Planner For Safe Navigation Amidst Moving Agents". In: (2020). arXiv: 1907.01577 [cs.RO].

[2] Ali Alizadeh et al. "Automated Lane Change Decision Making using Deep Reinforcement Learning in Dynamic and Uncertain Highway Environment". In: (2019). arXiv: 1909.11538 [cs.RO].