Report Autonomous Vehicle Driving

Group 06

Cirillo Benedetto[1], D'Amato Stefano[2], Spremulli Michele[3], Casaburi Adolfo[4]

[1,2,3,4]**{b.cirillo6, s.damato16, m.spremulli1, a.casaburi35}@studenti.unisa.it**

Autonomous Vehicle Driving

2023/2024

Contents

1. Introduction

The purpose of this project is to deisgn, implement and evaluate an autonomous driving system that respects as much as possible the road rules.
The realization of the project is made possible through the utilization of the simulator CARLA, starting from a baseline of reference. The baseline already implements the main components of an autonomous driving system, that are the global planner, the behavior planner, the local planner, along with PID controller for longitudinal control and Stanley controller for lateral control.
Our goal was to analyze the main features of the baseline, highlighting critical issues and trying to improve them, integrating the management of some situations or behaviors that the baseline doesn't handle or handles badly.

2. Routes Description

The system of autonomous driving is applied on a car and tested in five routes, each of which with its weather conditions, environmental context and road scenarios. Follows a brief description of each route (table 1).

Routes	**Considtions**		
	Description	**Illumination**	**Weather**
Route 0	Curved road route, rural landscape	Transition from conditions of good lighting conditions to poor lighting, due to fall of the night	Cloudy weather conditions
Route 1	Curved road route, rural landscape	Transition from conditions of good lighting conditions to poor lighting, due to bad weather	Transition by weather conditions clear to cloudy, with presence of rain
Route 2	Urban route focused on high density areas with obstacles	Daily light, good illumination	Clear weather
Route 3	Urban route focused on high density areas with obstacles	Daily light, good illumination	Clear weather
Route 4	Rural route focused on crossing actors	Daily light, good illumination	Cloudy weather

Table 1. Routes Description

Each route consists of a series of events/scenarios that test the robustness of the autonomous driving system. Follows an overview of all the events that occurs in the routes (table 2).

Event	Description	Route
BlockedIntersection	The intersection is blocked by a stationary car	0 - 1
AccidentTwoWays	An accident is simulated. The scenario involves 3 vehicles on the side of the road. Vehicles remains stationary, so they must be overtaken to continue the route	1 - 4
HazardAtSideLaneTwoWays	A couple of cyclists are driving on the lane, stopping at a certain point. The autonomous vehicle must overtake them in order to keep on going	0 - 1
InvadingTurn	Vehicles coming from opposite lane partially invade the lane of ego vehicle, due to a lane narrowing	0 - 1
ParkedObstacleTwoWays	On the laneside thereis a parked car to be overtaken	0 - 1
HardBreakRoute	The leading car suddenly stops and remain stationary for a while. After a brief pause, it will then start moving again	0 - 1
ControlLoss	The scenario involves the car skidding due to irregularities in the road surface	1
DynamicObjectCrossing	A pedestrian suddenly crosses the lane	4
NonSignalizedJunctionRightTurn	The autonomous vehicle must face an intersection not properly signalized	4
VehicleTurningRoutePedestrian	The autonomous vehicle properly faces a turn, but before completing it, there is a pedestrian crossing the road	4
VehicleTurningRoute	The autonomous vehicle encounters a cyclist after taking a turn	4
ConstructionObstacleTwoWays	The road is occupied by a traffic warning signal that makes impossible for the car to drive along the lane	0 - 1

Table 2. Events/Scenarios description

3. Baseline Analysis

As already said, the project starts from a baseline that already implements the main components that an autonomous driving system needs, that are:

- **mission planner**, able to provide the mission planning, in order to properly navigate each route, finding the route through the A* algorithm;
- **behavior planner**, that implements a *rule based* approach for managing different situation, but not all that a vehicle can found on the road. The hierarchy of the rules is the following:
  1. traffic light management - takes effect when the ego vehicle encounters a red traffic light;
  2. pedestrian avoidance - allows the ego vehicle to stop when it is too close to a pedestrian;
  3. vehicle avoidance - allows the ego vehicle to stop when it is too close to a vehicle;
  4. car following - the ego vehicle follows the leading vehicle if it is at a safety distance, otherwise it starts decelerate;
  5. intersection behavior - takes effect in proximity of intersections, slowing down the vehicle, but doesn't check for other vehicles invading the intersection;
  6. normal behavior - the ego vehicle follows the preset max speed or the speed limit when none of the listed situations happen;
- **local planner**, that provides the interface to set a target speed or to modify the current path, setting a new global plan or adding new waypoint to the existing queue;
- **controllers**, a PID for longitudinal control and a Stanley Controller for lateral control.

3.1. Evaluation Metrics

To evaluate the autonomous driving system, a series of infraction with the corresponding **penalty** term has been established; the product of this infraction penalties are then combined with the percentage of **route completion**, forming the main metrics: the **driving score**. The average of each of this three evaluation metric is computed when all routes have been completed. They are named *Global metrics*. **Global driving score** is the main metric.

3.2. Baseline performance

The simulation of the autonomous driving system "guided" by the baseline is largely insufficient with respect to the evaluation metrics and highlights critical issues in managing the most recurring situations on the road.

	Route0	**Route1**	**Route2**	**Route3**	**Route4**	**Global**
Evaluation metrics						
Driving Score	8.6e-03	2.154	4.73	9.48	23.07	7.89
Route Completion	100	12.14	4.73	9.48	55.34	36.34
Infraction Penalty	7.6e-05	0.177	1	1	0.42	0.52
Infractions						
Number of occurrences	23	5	1	1	4	34
Collision with pedestrian (0.5)	2	0	0	0	0	2
Collision with other vehicles (0.6)	10	1	0	0	1	12
Collision with static elements (0.65)	2	2	0	0	0	4
Running a red light (0.7)	0	0	0	0	0	0
Running a stop sign (0.8)	1	0	0	0	0	1
Scenario Timeout (0.7)	5	1	0	0	1	7
Min speed	3	0	0	0	1	4
Outside route lanes (0)	0	0	0	0	0	0
Route deviation (0)	0	0	0	1	0	1
Vehicle Blocked (0)	0	0	1	0	1	2

Table 3. Baseline performance for each route

3.3. Considerations

It's clear from the table 3 that the baseline is unable to handle the most of the route, both for infraction penalty and for the non-completion of them.

Let's see more in detail the composition of infractions (figure 1): based on this analysis, we can draw up a classification of the most impactful events on the evaluation.

1. **Collision with other vehicle** happen mostly in the first and second route but strongly impact the overall evaluation, since many situations of possible collision are not contemplated by the baseline. Among them there are:

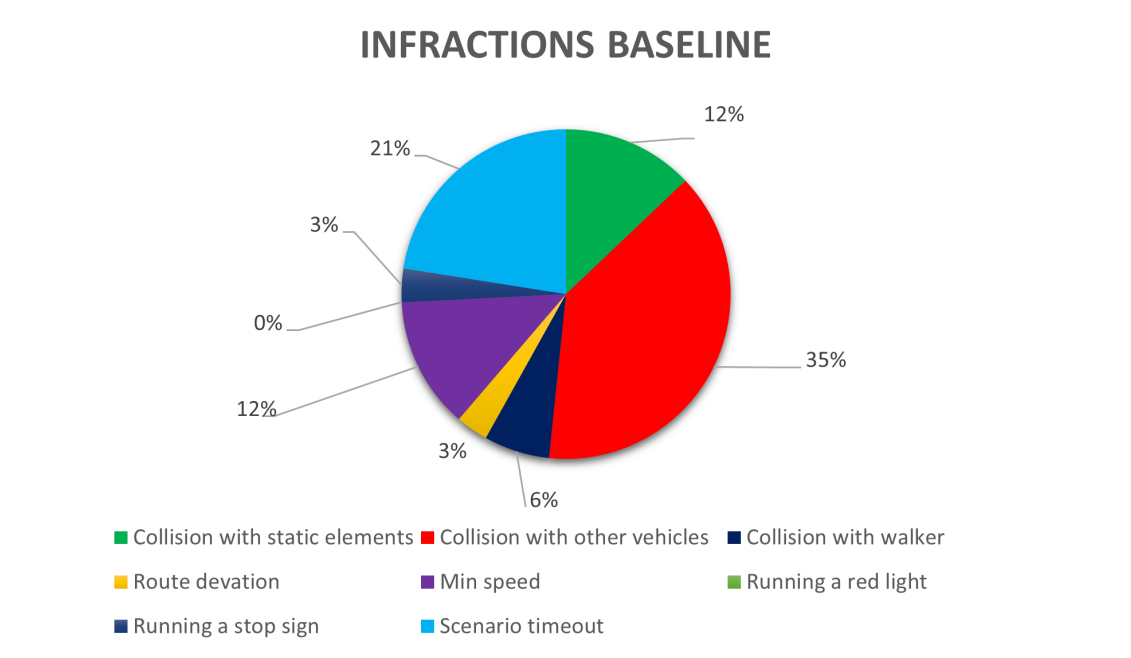

Figure 1. Infractions composition

- *parked vehicle*, in this case the baseline turns out to be lacking in overtaking logic and in detection logic of this particular kind of vehicles;
- *narrowing of the roadway*, that causes a collision with vehicles coming from the opposite lane that partially invade the ego vehicle lane;
- *intersections not handles by a traffic light*, the ego vehicle is not able to detect the stop sign and occupies the intersection without worry about other vehicles;
- *dangerous turns that presents a stationary vehicle immediately afterwards*, due to a lack of speed control in turns, the ego vehicle is too fast to detect them.

2. **Scenario timeout** is an event that occurs when the ego vehicle is unable to overcome a given scenario within a specific time limit. This can cause the interruption of the route, impacting both in infraction penalty and in percentage of completion.

3. **Collision with static elements** happens since the baseline is unable to detect traffic warning signals along the roadway.

4. Regarding the **min speed** issue, in the real world no "min speed" infraction is formally defined in the contexts where the routes are set, so we decided not to take into account this aspect.

5. **Collision with pedestrian** happens most of the time since the pedestrian is treated almost as a car in the baseline, while much more attention must be put in these situation, cause it is the most penalizing metric.

6. **Running a stop route** happen since there is a lack of logic for the stop signal management. It is related to the infraction of collision with other vehicles, cause a stop signal is placed in proximity of intersections.

In order to improve the baseline, we followed this classification since it is the shortest way to achieve a better performance. The last consideration is that, since almost all the route are not complete due to the scenario timeout event, it can be possible other event that are not contemplated in this initial analysis, so, in this cases, we dealt with events as they came our way.

3.4. Operational Design Domain

Finally, the Operational Design Domain (ODD) is presented to summarize the functionalities of the baseline and the contexts in which it provides a good solution.

Physical infrastructure	Roadway types	The system is able to travel in urban roads with one or two lanes and in suburban roads with one lane. In sharp curves sometimes invades the adjacent lane.
	Fixed road structures	The system is able to detect and respect traffic lights. It doesn't respect traffic rules in intersections.
	Temporary road structure	The system is not able to detect temporary static elements on the road, such as traffic warnings and cars parked at the roadside. In this scenarios, it causes a collision.
Operational constraints	Speed limits	The system is able to retrive the speed limit of the current road and stay below that value.
	Traffic conditions	The system is able to navigate in no or moderate traffic conditions. It doesn't implement a logic for the overtake, so when the road is blocked by some elements, the system stays stationary.
Environmental conditions	Illumination	The system can navigate both during the day, in good light condition, and during the night, with poor light conditions.
	Weather	The system can navigate in cloudy or wet weather; in presence of rain, when performing an emergency stop, the system lose the control.
Dynamic elements	Pedestrian crossing	The system badly manages situations in which a pedestrian suddenly crosses the road not in a pedestrian crossing point, investing the pedestrian.

4. Improvements

In this section will be presented and analyzed the improvements that have been made to the baseline, starting with the calibration of the two controllers followed by an analysis of the most impacting events, presented in the previous section, that allowed us to modify the behavior of the vehicle.
We adopted a *Normal* styles of guide, among the three already provided by the guideline (the other are *Cautious* and *Aggressive*). This kind of style of guide already provides some fixed measures which pertaining to that type of guide, such as the *target_velocity=50(km/h)*, *breaking_distance=4(m)* or the angular opening that defines the field of view of the ego vehicle during detection of other vehicles $[+60, -60]$, or for the detection of pedestrian $[+90, -90]$ (considering 0 the direction of the ego vehicle). While some others parameters have been introduced to optimize the new functionalities, such as the $safety\_distance = (ego\_vehicle\_speed(km/h)/10)^2$ [1] (where *ego_vehicle_speed* is the current speed of the ego vehicle), involved in many modules.

4.1. Calibration of controllers

Lateral and longitudinal controllers are the core of the vehicle stability, and on them rely all the other components of an autonomous driving system, so we decided to start with their calibration in order to obtain a robust control over the vehicle.
The calibration of controllers has been conducted on empty route with just two turns (one to the left and one to the right), just to understand how the vehicle responds to certain inputs, without being affected by other situations.
Since the simulation context is of type *fixed time_step* with $fixed\_delta\_second = 0.05$, we set dt of both lateral and longitudinal controller to this value.

4.1.1. Longitudinal Control

The longitudinal control is actuated through a *PID Controller*, for which the baseline provides the following values (table 4):

K_P	K_I	K_D	dt
1.95	0.05	0.2	0.05

Table 4. PID Controller parameters for the baseline

With this configuration the speed has the following trend (figure 2):

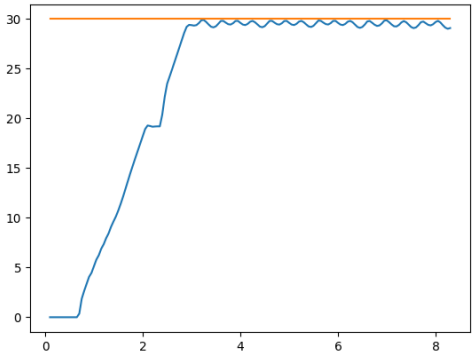

Figure 2. Speed trend

We considered the rise time of little significance, since we don't aspire to a very reactive system

in acceleration; it is much more relevant the steady-state error, that appears to be oscillating. So we increased the K_I and decreased the K_P in order to balance the growth of the overshoot. With this first changing, we obtained an acceptable overshoot and settling time, while the steady-state error and the rise time could be even improve by decreasing K_D, whose change generates a little overshoot, but still acceptable, but eliminate the oscillation . The final in configuration is the following (table 5):

K_P	K_I	K_D	dt
0.8	1.0	0.07	0.05

Table 5. PID Controller parameters for the final system

This configuration allows the speed trend (figure 3) to have a much better trend, and it allows a more comfortable driving style.

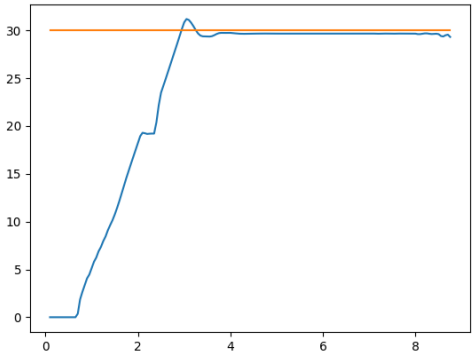

Figure 3. Speed trend

4.1.2. Laterl Control

Lateral control is implemented with the *Stanley Controller* with the following values in the baseline (table 6):

K_V	K_S	dt
1	0.01	0.05

Table 6. Stanley Controller parameters for the baseline

The calibration of these parameters was quite more complicated then the PID's ones, since we could not prove it in a separate route, since in order to obtain a robust control we had to prove it in different kind of turns and in different situations, such as the overtake and sharp curves. So the process which led to the parameters' establishment has been iterative and involved many trials on different scenarios. Are then reported the final values (table 7).

K_V	K_S	dt
1.8	0.01	0.05

Table 7. Stanley Controller parameters for the finale system

4.2. Collision with other vehicles

This kind of infraction is the most impactful on the overall score, so we started our optimization by analyzing when and why this situation occurs, trying to avoid collision as much as possible. Collision turns out to happen in different situation, listed in the previous paragraph, that are now analyzed more deeply and a feasible solution is presented.
Most of these situations involve an overtake to be overcome; due to its complexity it is discussed in a separate section: 5.

4.2.1. Parked vehicles

The events *AccidentTwoWays* and *ParkedObstacleTwoWays* present some cars parked at the roadside. The baseline generates a collision with this kind of vehicles, since the detected lane id is different from the ego vehicle's one, so it simply doesn't care about them, considering them on the adjacent lane. But, since they occupy about half of the lane, they have to be taken into account. In order to consider them we extended the function *vehicle_obstacle_detected*, already implemented in the behavior agent: now all the vehicles on the adjacent lane at the right of ego vehicle's direction of travel that have speed equal to 0, must be overtake. It is also important to check that the ego vehicle is on the lane further to the right of the road, otherwise, in a road with two lanes for direction of travel, it should overtake vehicles in the lane on its right that are queuing at a traffic light, for example, and have speed 0.

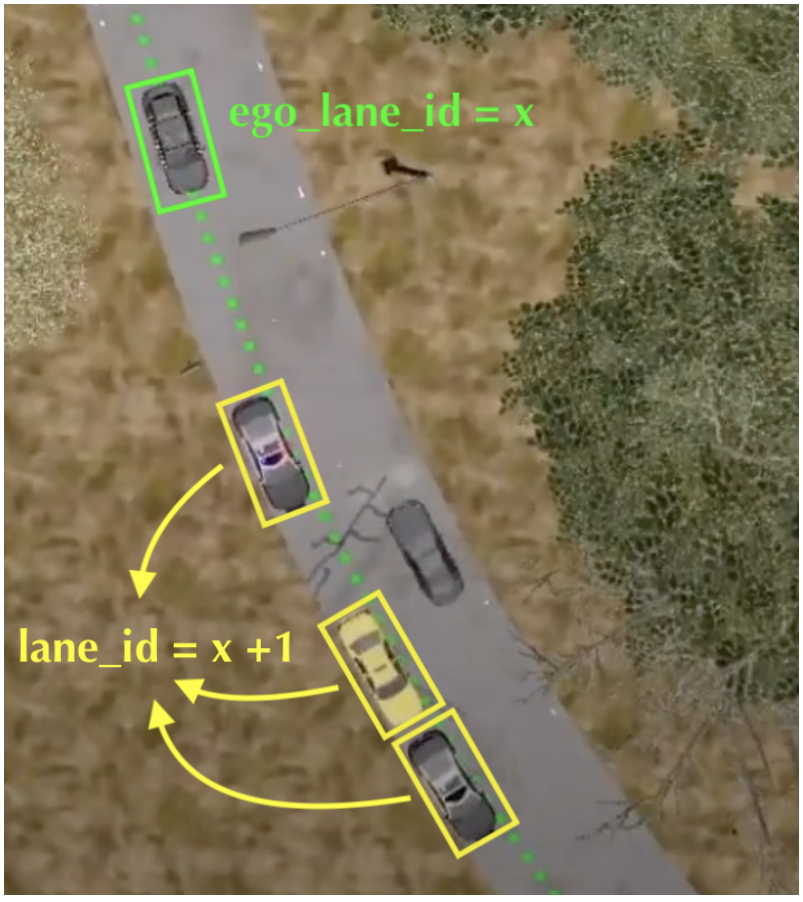

Figure 4. Vehicles parked at the roadside are considered on the adjacent lane

4.2.2. Roadway narrowing

This kind of situation causes the vehicle coming from the opposite lane to partially invade the lane on which the ego vehicle is travelling and generates a collision.
In order to avoid this behavior, along with the existing rules that the vehicle has to follow described in the baseline analysis (3), it also must check, for each coming vehicle at a distance less then 20 meters, how much it is misaligned from the center of its lane; consequently the lateral controller sets an offset from the center of lane towards the edge of the road equal to the twice this value, if the misalignment is towards the center of the lane (in direction of the ego vehicle). In case of misalignment grater the 0.5 meters, the target velocity is decreased to 20 km/h, to allow the performing of the deviation in time. (Figure 5)

Figure 5. Lane invasion management due to the scenario *InvadingTurn*

4.2.3. Intersection with no traffic light

If an intersection is not handled by a traffic light, it surely has a stop signal for the merging roads. The baseline doesn't detect this situation, causing two infractions: *running a stop sign* and, probably, *collision with another vehicle*, since it doesn't follow priority rules dictated by road signs. Due to its complexity, the implemented behavior in this kind of situation is discussed in a separate section 6.

4.2.4. Dangerous turns

After some turns there may be a stationary car that the ego vehicle cannot detect in time to stop and generates a collision.
This can be avoided simply slowing down in proximity of turn. This situation is configured when the following condition is verified: thanks to the method *get_ incoming_ waypoints* implemented in the local planner, we can obtain the queue of the next n waypoints, with n calculated as the current velocity divided by 10; then we calculate the angle between the direction of each waypoint in the queue and the ego vehicle, using the *carla.Location* attribute of *carla.Wayoint* object. At the first angle that turns out to be greater then 5.5 degrees, we can assert that we will face a turn. In this case the target speed is decreased to 20 km/h.
There is an anomalous situation in Route1, in which the other vehicle is turned upside down causing a collision. This happens because the field of view used to check vehicles in the proximity of the ego vehicle (in normal condition) is set in the range $[+60, -60]$. This range is used in the method *is_ within_ distance* that checks if the angle between the forward vector of the vehicle and the forward vector of the ego vehicle is within this range. In this case, due to the "anomalous" orientation of the vehicle, the angle is outside the range, as shown in figure 6. To manage this situation, if the angle between the two forward vectors is higher then 90 we set the range to $[90, (90 + 60)] \cup [-90, -(90 + 60)]$ (figure 7.

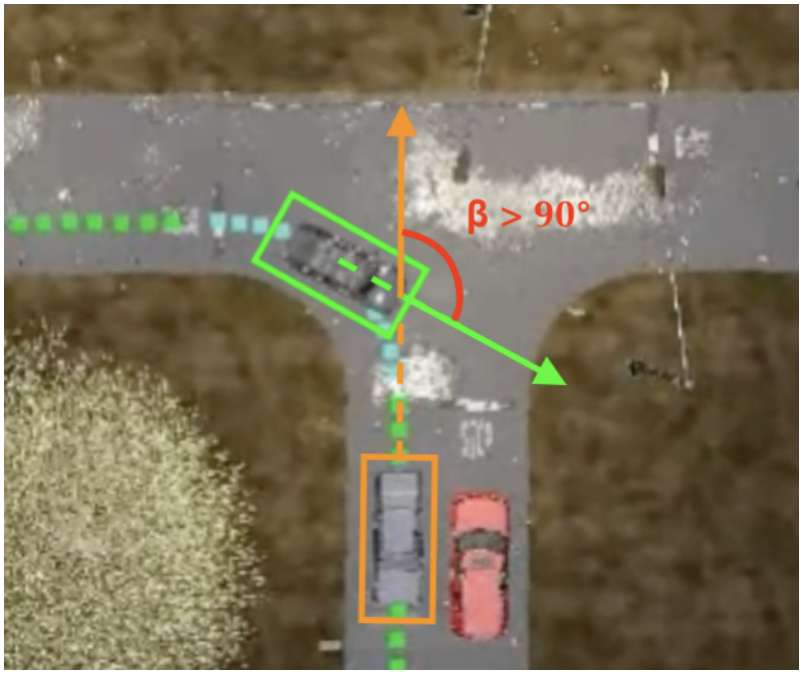

Figure 6. Anomalous situation: vehicle turned upside down

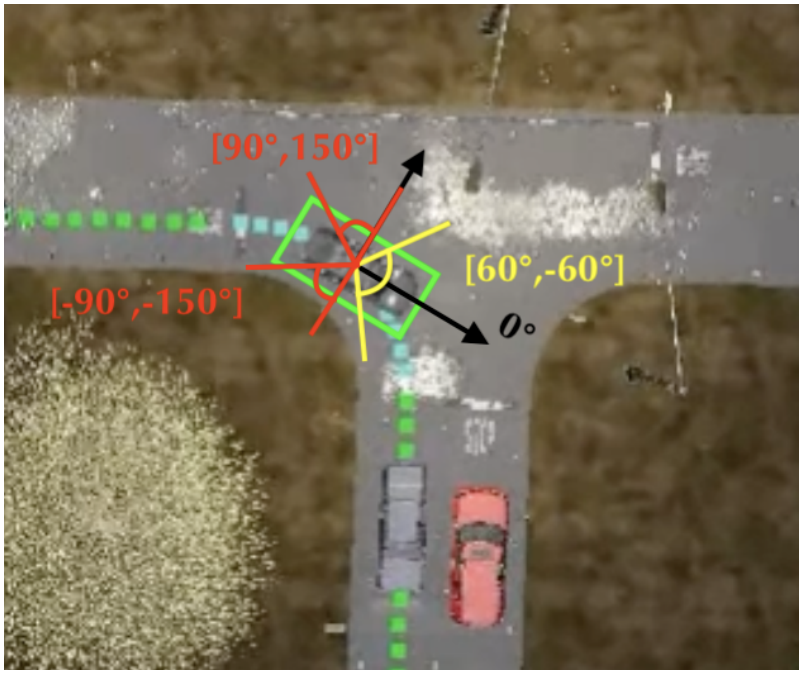

Figure 7. normal field of view (yellow) and field of view for anomalous situation (red)

4.3. Collision with static elements

The collision with a static object always happen in the scenario *ConstructionObstacleTwoWays*, when a traffic warning (figure 8) makes the travelling on the lane unfeasible. The baseline rules don't include the management of such a situation, so we extended the method *collision_and_car_avoid_manager* of the behavior agent in order to include in the list of elements to detect also the static ones, so they are consider as other vehicles and the logic of

Figure 8. Traffic warning

car_following_manager is applied likewise. What distinguishes the static objects from other vehicles on the same lane of the ego vehicles is the overtaking: while the ego vehicle can't overtake a stationary vehicle on its same lane, since it may be stopped for a lot of reasons, static element must be overtake.

4.4. Collision with pedestrian

This is the most penalizing infraction, so we put so much effort to avoid it as much as possible. The two scenarios in which a pedestrian invades the road are *VehicleTurningRoutePedestrian* and *DynamicObjectCrossing* (figure 9): in the first one the pedestrian crosses the road when the ego vehicle is finishing the turn in an intersection, while in the second one it comes out suddenly from a roadside element; both cases are difficult to manage because, when a pedestrian step on the road, there may be too little space from him and the ego vehicle to perform a safety stop. Since they are pretty similar situations, one and the same logic is applied to handle them.
The baseline already provide the management of pedestrian, forcing an emergency stop when they

Figure 9. pedestrian comes out suddenly from a roadside element

are at a distance less then a fixed breaking distance; it turns out to be too little for an emergency stop and the pedestrian is invested. The problem is that, in phase of detection, the pedestrian is treated as a vehicle, so the checks for the road and lane id are equally applied. The only difference is the angle of the fields of view among the two situations: for the car it is set to $[+60, -60]$, while for pedestrian it is $[+90, -90]$ (figure 10). This is not a good solution since vehicles, unlike pedestrians, almost always have a predictable behave: we will never find vehicle crossing the road like a pedestrian (hopefully).

Our solution doesn't take into account the lane id and the road id in which the pedestrian is located, but only the distance and the angle to which it is detected. This logic is applied in the same method called for the detection of vehicles and static objects (*_vehicle_obstacle_detected*), that has a new parameter identifying when it is called for the detection of vehicles/static objects or for pedestrians; in the latter case it doesn't check for lane and road id, in order to impose less constraints to the detection and be more safe. Also an emergency stop with an higher brake brake then usual (1 instead of 0.75), to make the stop stronger.
The downside of this logic is that every time a pedestrian is detected at the roadside, the ego vehicle stops even if he doesn't cross the road; this happens for a fixed amount of time, after which the ego vehicle slowly gets closer to the pedestrian (setting momentarily the breaking distance to 0), and, when it is no more detected, because he is no longer in the field of view $[+90, -90]$, the ego vehicle proceeds normally.

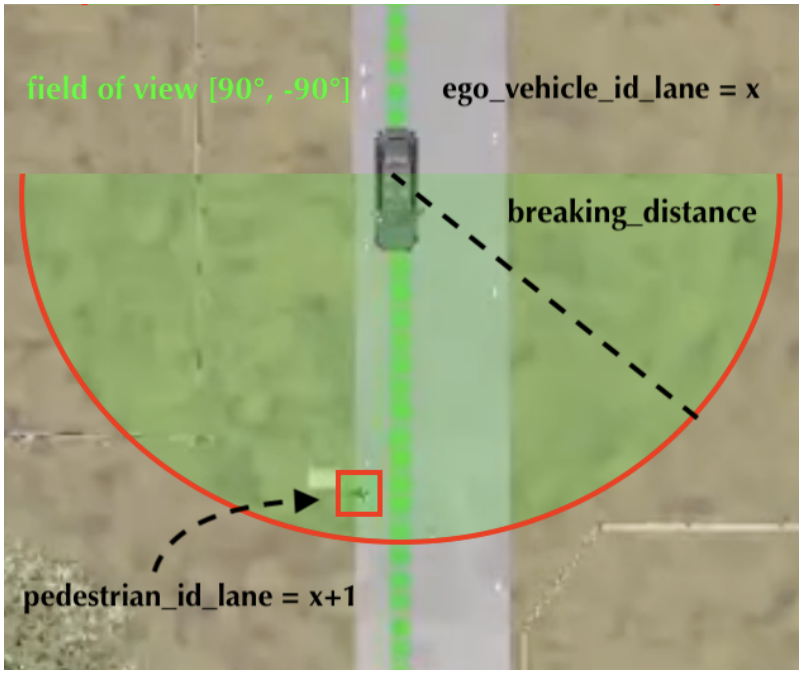

Figure 10. Field of view for pedestrian

5. Overtake

The baseline did not include any overtaking management logic, so we developed it from scratch. One of the most frequent causes of vehicle collisions is cars crashing into stationary object, parked vehicles and riders. After examining all the scenarios, the following ones were identified as the most detrimental and deserving of significant penalties:

- **AccidentTwoWays** and **ParkedObstacleTwoWays**: Each occurrence of these events causes a collision with at least a vehicle in the vehicle's normal route.
- **HazardAtSideLaneTwoWays**: In each occurrence of this event the ego vehicle collides with at least a bike or a motorcycle along its normal route.
- **ConstructionObstacleTwoWays**: In each occurrence the ego vehicle collides with the traffic warning signal.

As shown in the table 2, these scenarios happen many times and the absence of an overtaking management causes collisions or the "*scenario timeout*" so it must be developed to complete the routes with a high score.

First, we need a model that describes overtaking behavior and we choose the **uniformly accelerated motion** for the ego vehicle and **uniformly motion** for the others.

The implementation of the overtaking behavior follows the following steps:

1. overtaking distance's estimation
2. overtaking conditions's verification
3. overtaking path's generation

The list of objects (located in our lane) to be overtaken is:

- cars that are on the roadside
- bicycles
- motorcycles
- static objects

5.1. Physical Model

In order to use the uniformly accelerated motion model on our vehicle the **acceleration** of the vehicle is a fundamental piece and since we did not have immediate access to this parameter we had to *estimate it*. The experiments were led by performing 50 accelerations in the first route, observing the time taken by the vehicle to go from 0 to 30 km/h. The following table lists the experiments performed (table 8):

The table shows that the vehicle can generate an acceleration of approximately 3.047 m/s^2. After many attempts we decided to set $a = 3\ m/s^2$, to get a little margin.

Under this condition and knowing the total overtake distance, the time to complete the overtake is computed by solving the equation $s = s_0 + v_0 t + \frac{1}{2}at^2 \rightarrow t^2 + 2\frac{v_0}{a} - 2\frac{\Delta s}{a} = 0$ (discarding the minus solution as it is physically unacceptable).

experiment	v_0 (km/h)	v_f (km/h)	Time (s)	a (m/s^2)
1	0	30.0	2.703	**3.082**
2	0	30.0	2.658	**3.134**
3	0	30.0	3.032	**2.747**
4	0	30.0	2.544	**3.274**
5	0	30.0	2.512	**3.316**
6	0	30.0	3.023	**2.755**
7	0	30.0	2.970	**2.805**
8	0	30.0	2.698	**3.088**
9	0	30.0	2.877	**2.895**
10	0	30.0	2.672	**3.117**
11	0	30.0	2.938	**2.836**
12	0	30.0	2.492	**3.343**
13	0	30.0	2.760	**3.018**
14	0	30.0	2.769	**3.008**
15	0	30.0	2.887	**2.885**
16	0	30.0	2.915	**2.858**
17	0	30.0	2.927	**2.846**
18	0	30.0	2.537	**3.283**
19	0	30.0	3.052	**2.729**
20	0	30.0	2.674	**3.116**
21	0	30.0	2.748	**3.031**
22	0	30.0	2.615	**3.186**
23	0	30.0	2.664	**3.127**
24	0	30.0	2.544	**3.274**
25	0	30.0	2.771	**3.007**
26	0	30.0	2.935	**2.838**
27	0	30.0	2.537	**3.283**
28	0	30.0	2.910	**2.862**
29	0	30.0	2.653	**3.139**
30	0	30.0	2.580	**3.228**
31	0	30.0	3.056	**2.726**
32	0	30.0	2.484	**3.354**
33	0	30.0	2.486	**3.351**
34	0	30.0	2.522	**3.302**
35	0	30.0	2.477	**3.363**
36	0	30.0	2.915	**2.858**
37	0	30.0	3.051	**2.730**
38	0	30.0	2.931	**2.842**
39	0	30.0	2.812	**2.962**
40	0	30.0	2.493	**3.342**
41	0	30.0	2.463	**3.383**
42	0	30.0	2.793	**2.983**
43	0	30.0	2.743	**3.037**
44	0	30.0	2.924	**2.849**
45	0	30.0	2.668	**3.122**
46	0	30.0	2.601	**3.202**
47	0	30.0	3.015	**2.762**
48	0	30.0	2.561	**3.252**
49	0	30.0	2.996	**2.780**
50	0	30.0	2.708	**3.077**

Table 8. List of experiments for estimating acceleration

5.2. Overtaking distance

The distance *overtake_d* to be traveled during overtaking was calculated as follows:

$$\begin{cases} overtake\_d = other\_line\_d - 2 * lane\_change\_d + 2 * hypotenuse\_change\_d \\ other\_line\_d = distance\_from\_last\_obj + obj\_length/2 + safety\_space\_reentry \end{cases}$$

The total distance **overtake_d** is computed (as shown in the figure 12) as the distance that the ego vehicle would have to travel in the other lane (**other_line_d**) to pass the obstacle(s), also considering the oblique sections of movement between the lanes (- 2**lane_change_d* + 2**hypotenuse_change_d*), so that we can more accurately calculate the true distance traveled by the vehicle (lane_change_d is an hyper parameter set to 2.75 meters after several attempts) . *other_line_d* is computed as the sum of 3 terms: the distance to the last object to be overtaken (**distance_from_last_obj** computed in this algorithm 1), half the length of the last object to be overtaken and a margin of space to safely get back into the lane $safety\_space\_reentry = 4\ m$ as shown in the figure 11.

Algorithm 1: The algorithm for computing the distance to the last object to overtake

```
Data: ob_list, safety_distance_for_reentry
Return: distance_from_last_obj;
search_for_reentry ← True;
i ← 0;
while search _for_ reentry do
    if i == len(ob_list) − 1 then
        search_for_reentry ← False;
    else
        ob1_length ← ob_list[i].length/2;
        ob2_length ← ob_list[i + 1].length/2;
        dist ← distance(ob_list[i], ob_list[i + 1]);
        distance_between_objects ← dist − (ob1_length + ob2_length);
        if distance_between_objects > safety_distance_for_reentry then
            search_for_reentry ← True;
        else
            i+ = 1;
        end
    end
end
distance_from_last_obj ← distance(ob_list[i],ego_vehicle)
```

The list of possible objects to overtake **ob_list** is sorted by distance from the ego vehicle in ascending order. To find the last object to overtake we check the distance between adjacent pairs in *ob_list* and check if there is space to get back into our lane (*distance_between_objects* > *safety_distance_for_reentry*). The function **distance** compute the distance between the two centers of the objects passed as input. **safety_distance_for_reentry** is equal to *ego_vehicle_length* + *safety_space_reentry*.

These images (11 and 12) are used to make the variables considered in calculating overtaking distance as clear as possible.

5.3. Overtaking feasibility

Once the distance and time required for the overtaking maneuver has been defined in a safe way, it must be considered whether it is possible to overtake without:

- colliding with objects to be overtaken
- colliding with vehicles coming from the opposite lane

Figure 11. Visualization of other_line_distance (in this case $distance\_between\_objects > safety\_distance\_for\_reentry$

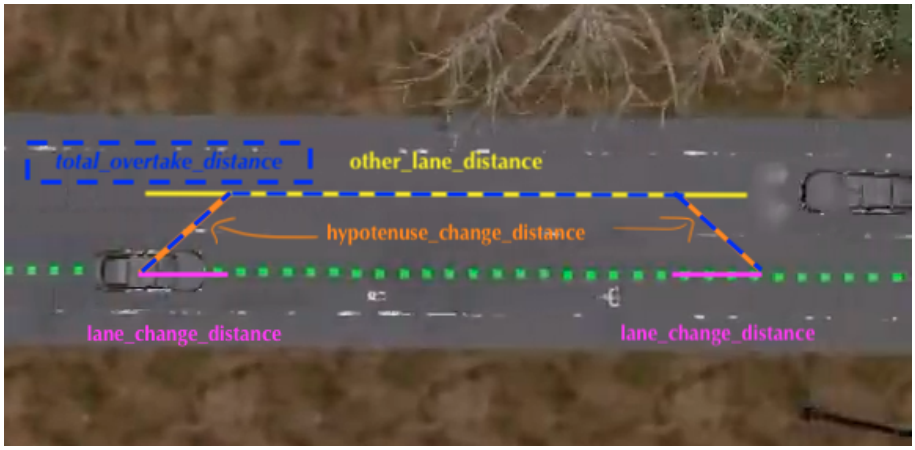

Figure 12. Visualization of total_overtake_distance

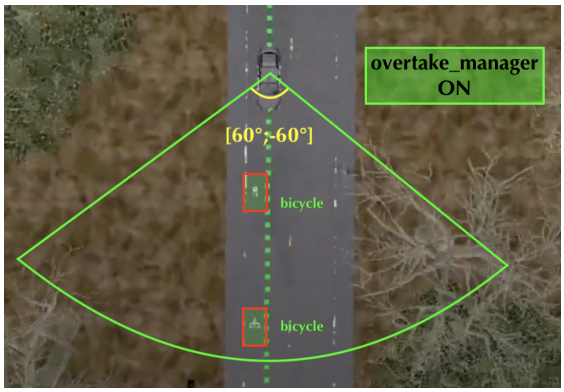

Figure 13

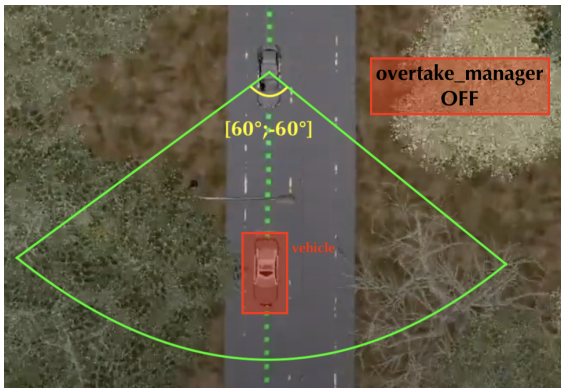

Figure 14

In Algorithm 2 there is pseudo code describing the algorithm for checking overtaking feasibility.

The method **overtake_manager** is called when is reported the presence of objects belonging to the list of elements described above (as shown in figure 13 and 14) that are at a *safety distance* from the ego vehicle.

The **overtake_manager** checks in a range of 150 m if there are vehicles in the opposite lane

Algorithm 2: The algorithm for check the feasibility of overtaking

```
Data: target_vehicle_distance, other_line_distance,
      target_vehicle_velocity,total_overtake_distance
search_for_reentry ← True;
i ← 0;
if target_vehicle_distance > other_line_distance then
    t_acc ← (v_max − v_0)/a;
    s_acc ← v_0 · t_acc + 0.5 · a · t_acc^2;
    target_vehicle_time ← (target_vehicle_distance -
     other_line_distance)/target_vehicle_velocity;
    if s_acc ≥ total_overtake_distance then
        other_line_time ← solve second degree eq as a function of time with s =
         total_overtake_distance;
        if target_vehicle_time > other_line_time then
            Return: True, other_line_distance;
        end
    end
else
    if t_acc < target_vehicle_time then
        t_const ← (total_overtake_distance − s_acc)/v_max;
        other_line_time ← t_acc + t_const;
        if target_vehicle_time > other_line_time then
            Return: True, other_line_distance;
        end
    end
end
Return: False, 0;
```

with a 30° angle of view or 90° if approaching a curve (see 4.2.4 to understand when there will be a curve). Once the vehicle is detected we assume, as told above, that it follow a uniform rectilinear motion with the detected velocity (after several experiments, we noticed that vehicles in the other lane were not observe the speed limit, so we set the velocity to 70 km/h if the detected velocity was below this value).

A necessary condition for overtaking is that the vehicles in the other lane is at a distance greater than the distance from the ego vehicle to the re-entry point (as shown in figure 15).

Then the algorithm checks if the space that the ego vehicle travels in uniformly accelerated motion (the one that travels from v_0 to v_{max}) is greater than **total_overtake_distance**. If so it means that the ego vehicle follows an accelerated motion for the whole overtaking maneuver and therefore the algorithm calculate the time that the vehicle takes to travel *total_overtake_distance* with the formula of the time in the uniform acceleration motion, and checks if it is less than **target_vehicle_time** (the time that the vehicle in the opposite lane takes to reach the overtaking return point). If so, **overtaking can be done** (figure 17), otherwise not (figure 16) . If the first condition is not respected, it means that the distance **total_overtake_distance** can be considered covered for a distance s_{acc} with accelerated motion and with a distance $d = total\_overtake\_distance - s_{acc}$ in uniform motion. Once the time taken to cover d in uniform motion has been found, the algorithm checks it the time $t_{acc} + t_{const}$ is less than *target_vehicle_time*. If so, **overtaking can be done** (figure 17), otherwise not (figure 16) .

When the vehicle is involved in an overtaking manoeuvre, it enters a new state (we set the variable *self._overtake* True) in which no other overtaking will be possible until the current one is completed. To do this, we check if the ego vehicle has consumed the waypoints added to make the overtake.

These images (15, 16 and 17) are used to make the variables considered in calculating overtaking

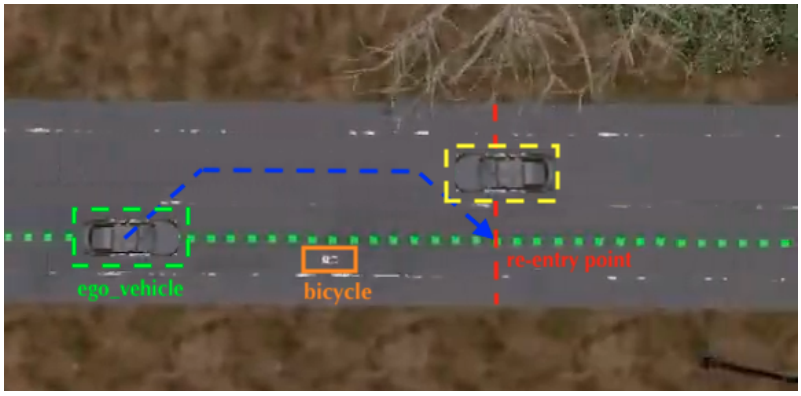

Figure 15. Unfeasible overtake

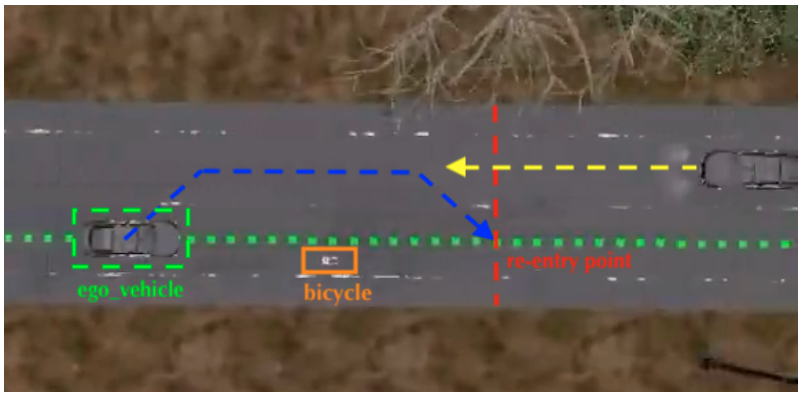

Figure 16. Unfeasible overtake

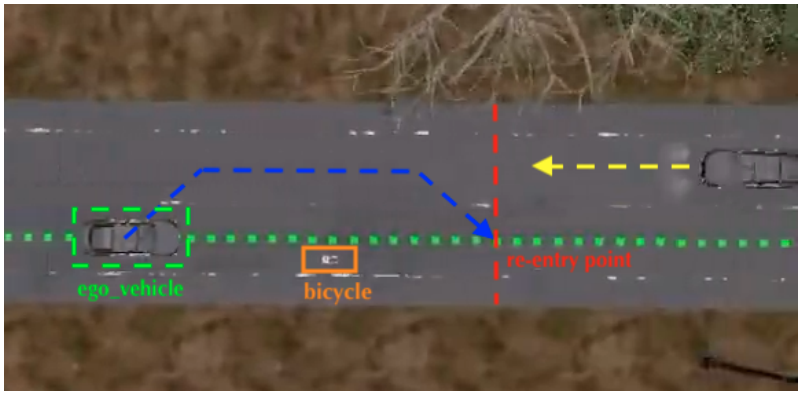

Figure 17. Feasible overtake

distance as clear as possible.

5.4. Overtake path generation

To generate the path (the waypoints) to follow during overtake we modified the **_generate_lane_change_path** method in this way:

- we add the possibility of generating the way points in the other lane in the opposite direction of travel (i.e. in agreement with our direction);
- we add the generation of the waypoints to return in our lane;

The input passed to the overtake are:

- *direction* = '**left**'
- *distance_ same_ lane* = 0 (we assume that the ego vehicle changes lane immediately)
- *distance_ other_ lane* = **other_ line_ distance -lane_ change_ distance**
- *lane_ change_ distance* = **lane_ change_ distance**

Then the generated waypoints are replaced in the global plan by replacing the first waypoints that cover the distance from the ego vehicle to the last generated way-point to implement the overtake.

6. Intersection management

6.1. Analysis

The baseline's intersection behavior is efficient only when it is regulated by a traffic light: in this situation the ego vehicle doesn't have to worry about other vehicles. But in presence of stop sign, it ignores them and priority rules of intersections, causing the infractions of running a stop sign and (probably) collision with other vehicles.

6.2. Stop sign management

A new method has been implemented (*stop_ manager*) for the management of stop signs. It is placed after the method that checks for traffic lights (see the baseline's behavior planner rules hierarchy at 3).

In figure we see the structure of the intersection that allows us to better understand the behavior to be adopted.

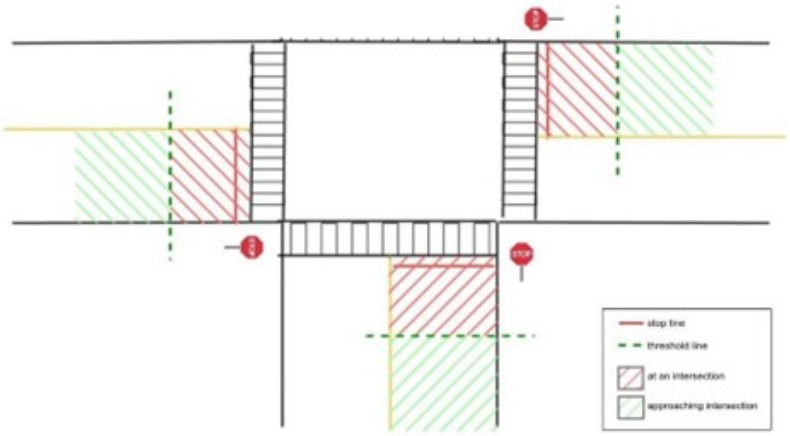

Figure 18. Intersection structure

This method detects stops signs within a distance defined by the *safety_ distance* parameter. When the stop is detected, the ego vehicle (entering the red barred area in the figure 18) starts braking, so as to stop near the stop sign. There is a penalty if the ego vehicle crosses a stop line with a speed higher than 0.1 m/s; in order to avoid penalties it has been established that the ego vehicle stops on the stop line for 10 seconds.

6.3. Detect lane obstacle

The method for managing intersections behavior is *detect_lane_obstacle*. This function involves the use of oriented rectangles, defined as bounding boxes oriented depending on the direction of the vehicle. All vehicles are initially detected at a distance of 40 meters from the ego vehicle. For each vehicle, the oriented rectangle is calculated, taking into account an extension factor: this is applied to predict a zone of points that will be occupied by the vehicle basing on its current velocity and direction of travelling. Then is detected a possible overlapping area between the area of the oriented rectangle of the vehicle and the area of the oriented rectangle of the ego vehicle. If there is no overlapping, then it can normally proceed in its path; otherwise the ego vehicle goes into emergency stop, until the overlapping area comes back to 0, that means that the vehicle is enough far from the intersection.

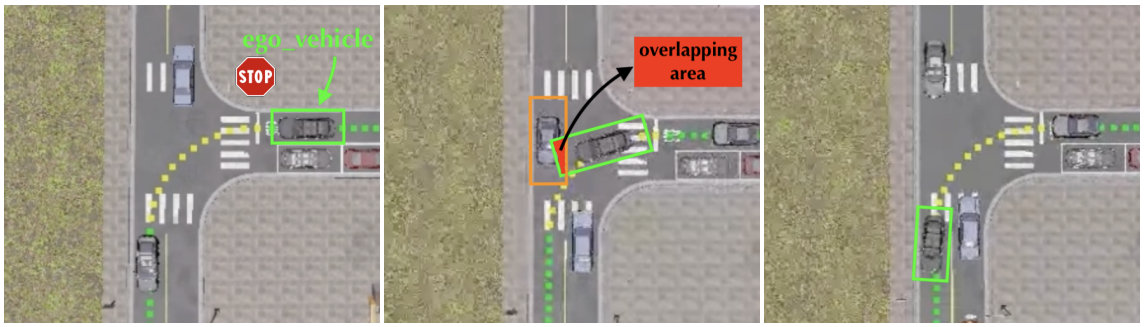

Figure 19. An example of the intersection behaviour

6.4. Intersection and traffic lights management

Another important aspect to consider is the simultaneous management of traffic lights and intersections. More precisely, if there is no traffic light at an intersection, the task of handling it is delegate to the *detect_lane_obstacle*; otherwise the intersection will be entirely managed by the traffic light.

7. Final System Analysis

As described in the previous sections, the major changes have been applied to the behavior planner thus changing the hierarchy of rules that handles the different situations. The new hierarchy of the rules is the following.

1. Roadway narrowing management - takes effect when the ego vehicle encounters a vehicle coming from the other lane invading the ego vehicle lane;
2. Traffic light management - takes effect when the ego vehicle encounters a red traffic light;
3. Stop management - takes effect when the ego vehicle encounters stop signal (described in section 6);
4. Pedestrian avoidance - allows the ego vehicle to stop when it is too close to a pedestrian (described in section 4.4);
5. Vehicle and static object management - takes effect when the ego vehicle encounter a vehicle (car, bicycle and motorcycle) or a static object. This point is further divided in three situations:

(a) Vehicle and static object avoidance - allows the ego vehicle to stop when it is too close to a vehicle or static object

(b) Vehicle following - the ego vehicle follows the leading vehicle if it is at a safety distance, otherwise it starts decelerate;

(c) Overtake management - takes effect when the ego vehicle encounters one of these objects: cars that are on the roadside, bicycle, motorcycle or static object (described in section 5);

6. Intersection behavior - takes effect when the ego vehicle is in proximity of intersections not regulated by a traffic light (described in section 6.2);

7. normal behavior - the ego vehicle follows the preset max speed or the speed limit when none of the listed situations happen.

7.1. Final System Performance

	Route0	**Route1**	**Route2**	**Route3**	**Route4**	**Global**
Evaluation metrics						
Driving Score	36	100	100	100	100	87.2
Route Completion	100	100	100	100	100	100
Infraction Penalty	0.36	1	1	1	1	0.87
Infractions						
Number of occurrences	4	3	3	0	1	11
Collision with walker (0.5)	0	0	0	0	0	0
Collision with other vehicles (0.6)	2	0	0	0	0	2
Collision with static elements (0.65)	0	0	0	0	0	0
Running a red light (0.7)	0	0	0	0	0	0
Running a stop sign (0.8)	0	0	0	0	0	0
Scenario Timeout (0.7)	0	0	0	0	0	0
Min speed	2	3	3	0	1	9
Outside route lanes (0)	0	0	0	0	0	0
Route deviation (0)	0	0	0	0	0	0
Vehicle Blocked (0)	0	0	0	0	0	0

Table 9. Final system performance for each route

From table 9 is evident that there are huge performance improvements between the baseline and our solution. The percentage increase on evaluation metrics is:

- 1006.03% for the **Global Driving Score** passing from 7.89% to 87.2%;
- 175.92% for the **Global Route Completion** passing from 36.34% to 100%;
- 74% for the **Global Infraction Penalty** passing from 0.5% to 0.87%;

The only infractions that are made belong to two categories: *Collision with other vehicles* and *Min speed*. *Min speed* infractions are caused by the inability to maintain a minimum speed.

However, these infractions are inevitable in overtaking situations, where the car must wait for the appropriate moment to perform the maneuver and then slow down, or at stop signs thereby violating the minimum speed limit, but it does not affect the driving score, as discussed before. Let's analyze the other infraction for each route:

- **Route 0:** there are 2 *Collision with other vehicles*. One happens when the ego vehicle tries to overtake two cyclists on a sharp turn by hitting the last one. The second happens due to a narrowing of the roadway in a curve.
- **Route 1,2,3,4**: no infractions.

7.2. Operational Design Domain

The ODD of the final version of the system is much more consistent then the initial one. Now the system can be considered efficient and safe in much more operational contexts.

Physical infrastructure	Roadway types	The system is able to travel in urban roads with one or two lanes and in suburban roads with one lane. It is able to keep the lane also in sharp curves, without invading another lane.
	Fixed road structures	The system is able to detect and respect fixed road structures, such as stop signs and traffic lights. It can handle intersections with no more then four intersection branches; it doesn't respect the right-of-way rule, but progresses in the intersection only if no other vehicle is detected in proximities.
	Temporary road structure	The system is able to detect temporary static elements on the road, such as traffic warnings and cars parked at the roadside. In this scenarios, it can perform an overtake maneuver once safety conditions have been assessed.
Operational constraints	Speed limits	The system is able to retrive the speed limit of the current road and stay below that value.
	Traffic conditions	The system is able to navigate in no or moderate traffic conditions. During the overtake maneuver the speed of the vehicles on the lane on which the system moves is considered constant, not less then 70 km/h. In the intersections the behavior of others vehicles, such as if they respect or not traffic rules or if they use or not turn signals lights, is not taken into account.
Environmental conditions	Illumination	The system can navigate both during the day, in good light condition, and during the night, with poor light conditions.
	Weather	The system can navigate in cloudy or wet weather, even with the presence of rain.
Dynamic elements	Pedestrian crossing	Thay system can manages situations in which a pedestrian suddenly crosses the road not in a pedestrian crossing point, even in a slippery road, where a too hard braking could cause loss of control of the vehicle.

Table 10. ODD for the final system

7.3. Future Improvements

The main goal was to improve the overall driving score by trying to reduce the number of infractions as much as possible, but improvements are still widely possible. While for some cases, such as overtaking, we tried to design a robust logic, according to our experiments; for some others, such as intersection management, a more accurate logic could be implemented.

7.3.1. Intersections management

For sure, an improvement that must be made is to provide a better logic for the intersection management, which at current state doesn't take into account the rules of precedence. This was definitely going to be the next improvement. Another improvement would have been the monitoring of the behavior of other vehicles through their status (e.g. by checking turn signals lights).

7.3.2. Overtaking management

We observed that the designed overtake logic is not very efficient in proximity of sharp turn (in fact the only overtake that causes an infraction is when the ego vehicle tries to overtake two cyclists in a sharp turn). In addition, we assumed that the vehicles on the other lane have a standard behavior (they follow an uniform motion). For these reason, some improvements to be made for the overtaking management are:

- more accurate control of all the parameters needed when the ego vehicle is in proximity of sharp turns;
- taking into account a more complete behavior of the vehicles in the other lane.

7.3.3. Roadway narrowing management

We observed that, if the roadway narrowing is in a sharp curve at the right, the designed management for this situation can't get the ego vehicle to move so that not to impact the vehicle on the other lane.

7.3.4. Driving styles

Another improvement that could be made is to adapt different driving styles according to the context (e.g. a driving style that is more aggressive in extra-urban areas and more cautious in urban areas)

8. Video

At this link you can see the behavior of the vehicle in the 5 mandatory routes (we apologize for the video quality of all the routes, especially route 1, but the computational resources available did not allow us to do better).

References

[1] https://www.aci.it/laci/sicurezza-stradale/le-cautele-nella-guida/distanza-di-sicurezza.html. learning revolution. IEEE Trans. Pattern Anal. Mach. Intell. 42(9), 2113–2132 (2019)