TÉCNICO LISBOA

Robotics: Report second lab
Topic: Autonomous Cars

Gaia Morillo	101499
João Pedro Andrade Gonçalves	85211
Marco Mussato	101412

Robotics Course, Instituto Supérior Tecnico

Prof. João Fernando Cardoso Silva Sequeira & Rafael André Alves Cordeiro

February 2022

Abstract

The second laboratory is mainly focused on the development of a software able to control a simulated autonomous car. This software will be used to simulate and control a car moving inside the IST Alameda campus. The principal blocks of the software are: guidance and control. Usually, a block for the navigation of the car has to be considered, but due to lack of time and human resources, it was not possible to implement it in this project. This report explains in details how the software was developed and all the assumptions taken during the development. A section that explains how to use the program is also presented.

1 Introduction

The aim of this assignment, as stated before, was to develop a simulation of an autonomous car operating inside a defined environment: the IST Alameda campus. In Figure 1 a simplified architecture for the autonomous car, that summarizes the work developed, can be seen.

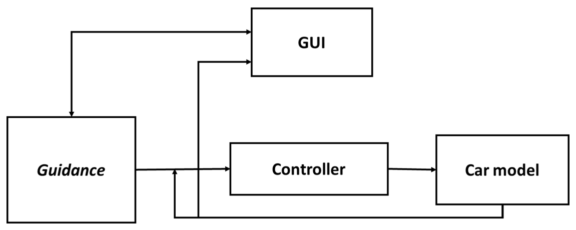


Figure 1 – *Block diagram of the simulator.*

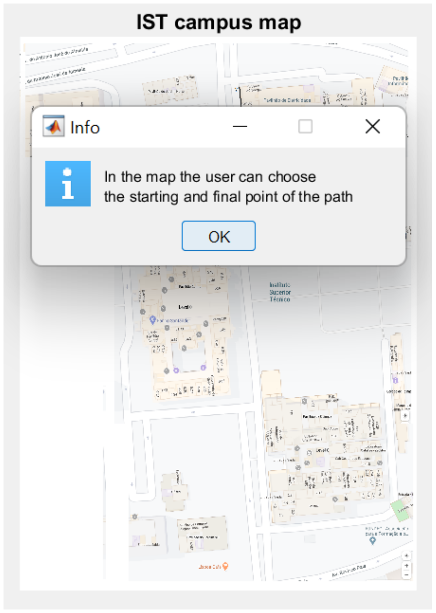
The guidance block empowers the car to generate an ideal, smooth and precise trajectory, between two specified points. This two points are the initial and ending points of the path and are decided by the user, using the simple GUI developed for the project.

The generated trajectory will be one of the inputs of the controller, whose task is to make the car follow the desired trajectory, trying to minimize the error between the real trajectory and the ideal one. The other input of the controller is the actual position of the car along time, that is computed inside the block that represents the car itself.

During the simulation, a simple representation of the car, moving along the path will be shown to the user and when the car reaches the final position a pop-up will show some relevant information related to the journey. In sections 3 and 4 a more detailed explanation of the mentioned block is presented, while the section 5 and the section 6 present a summary of the work and a summary of the future work needed to improve the project.

2 Ease of use

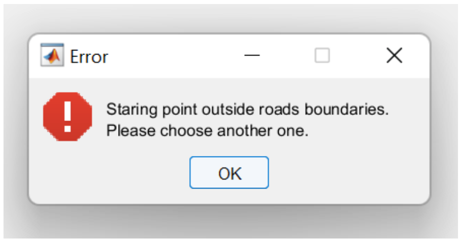
The software was developed in MATLAB and to run the code, the *Image Processing Toolbox*, has to be installed. Once this toolbox has been installed, the user just has to run the 'main.m' file, at which point an image with an overlapped pop-up, with a brief explanation, will be shown, as in Figure 2.



In the map the user can choose the starting and final point of the path

Figure 2 – *Example of the initial window.*

Clicking on the 'OK' button the pop-up will disappear and the user will be able to manually choose the initial and final point of the desired path, by clicking on them on the screen. Since the selected points must belong to a section of the map where there is a street, if the selected points are outside the allowed areas, an error message similar to the one in Figure 3 will be displayed.


Error

Staring point outside roads boundaries.
Please choose another one.

OK

Figure 3 – *Example of the warning presented to the user.*

If the points are inside the road, but it's impossible to compute a path between the two, another warning will be displayed and the user will have to start again with the selection of the points. This process will be iterated until a valid selection of points is done. Once the selection step is done, the initial map will disappear and a new window with the shortest path and the final trajectory will be shown, as in Figure 4.

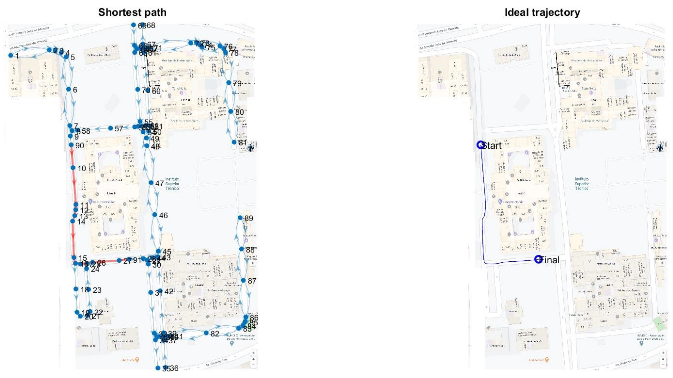


Figure 4 – *Example path and ideal trajectory.*

Afterwards, a new figure will be opened, Figure 5, showing two images. The one on the left will be a complete map of the campus, with a red dot moving on it, to show where the car actually is. The second one will be a zoomed view of the map, with the actual car position and orientation along time.

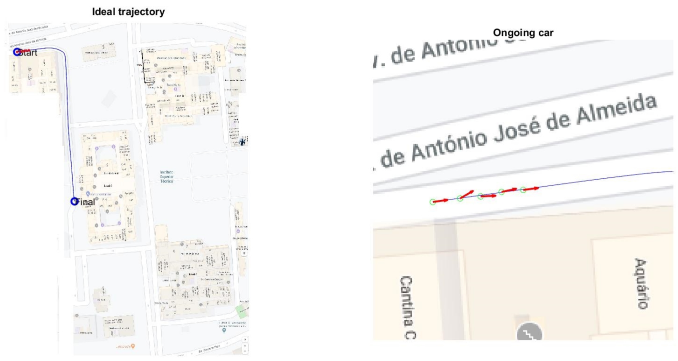


Figure 5 – *Image of the car moving.*

Once the autonomous car completes the itinerary, an informative pop-up will be displayed showing how much energy was consumed in the route, what distance the car traveled and how long it took. The last window shown to the user, once the previous pop-up is closed, will be the profile of velocity of the car during the route.

3 Guidance

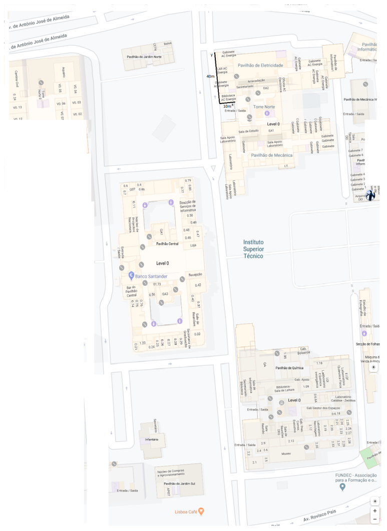


Figure 6 – *Complete map of the IST campus.*

The aim of the guidance block is to generate a reliable reference trajectory to be followed by the autonomous car. In this project, the trajectory is always generated inside a known environment, it being the map of the IST Alameda campus, shown in Figure 6.

3.1 Starting and final point

The process followed to obtain the final reference trajectory starts with the acquisition from the user of the starting and final point for the path. During this step, it is crucial to perform supervision of the points selected, in order to be sure that they are indeed 'selectable' points, so points that are belonging to roads of the campus. To do so, an occupancy matrix is being used, as shown in Figure 7.

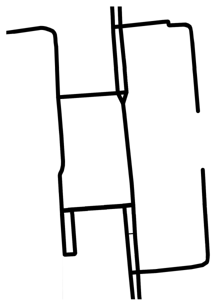


Figure 7 – *Occupancy matrix.*

If one of the selected points does not belong to a road point in the map, a warning will be displayed, Figure 3, and the user will be allowed to choose another valid point on the map.

3.2 Graph

Once the starting and final points are acquired from the user, they are used to upload the original graph of the campus. The original graph is a directional graph, whose nodes are in strategical positions along the roads of the campus and the direction of the edges between nodes are defined to follow the real direction of travel in each road inside the campus.

As shown in Figure 8, there are more points where there are curves or intersections and fewer points in straight parts of the roads. This disposition will be useful in the final trajectory generation, also to be able to have a more precise trajectory and a smooth velocity profile. This final concept will have a further explanation later on.

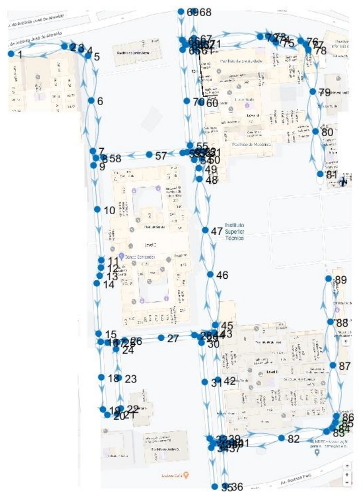


Figure 8 – *Directional graph.*

Moreover, each node is connected to specific weighted edges inside the graph. Each weight corresponds to the Euclidean distance between the two nodes connected by the edge. In this way, the distance between nodes is stored inside the graph and will be used as a parameter to define the shortest path between the starting and final points.

3.3 Graph updating

Once the user has selected two admissible points as starting and ending points for the trajectory, these points are used to update the graph. At this point, the distance between each edge of the graph and each of the two points is computed, to identify the right position for the points in the graph.

Afterwards, the new weighted edges are computed, and the graph is updated preserving the original direction of travel, as shown in Figure 9.

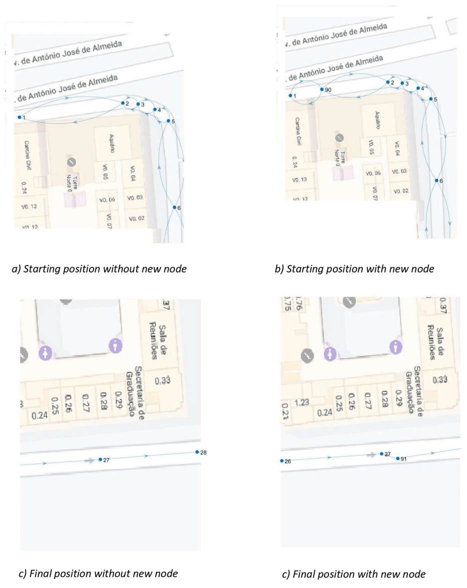

a) Starting position without new node
b) Starting position with new node
c) Final position without new node
c) Final position with new node

Figure 9 – *General procedure to update the graph.*

3.4 Path generation

After the previous step, the shortest path between the starting and final nodes is computed. To do so, a Dijkstra algorithm has been used, and since the weight of the edges are the distances between nodes in the graph, even if there are two possible paths with the same number of nodes, the one with the shortest distance will be chosen, as shown in Figure 10.

At this point, a further control for the path is performed. In fact, if the user had chosen two valid points (two points inside the road), but it's impossible to compute a trajectory between the two, a warning is displayed, and the user will be kindly invited to choose another pair for the starting and ending positions.

An example of this scenario could be that the only way to compute the path is to go in the wrong direction in a street. In this case a warning similar to Figure 3 will be displayed.

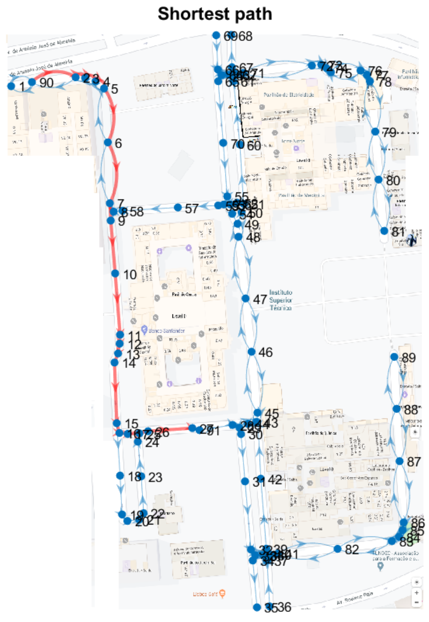


Figure 10 – *Example of path generation.*

3.5 Trajectory generation

After having computed the path, the program, using a process of interpolation, generates points between the selected nodes. For the interpolation we chose a value of 500 points per each node, which means between 2 consecutive nodes there will be 500 generated points.

Since, as mentioned before, the nodes in straight lines are further apart than the ones in curves, it is easy to identify where the car is turning or not. For that we compute the distance between points and, naturally, there will be a great discrepancy between the two situations, as shown in Figure 11.

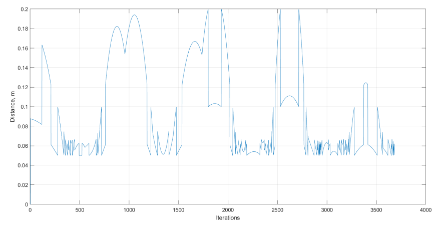


Figure 11 – *Distance between consecutive points.*

A threshold was computed doing the mean between the maximum and minimum value of the distances between points. That way we assume the car is in a straight line above the threshold and in a curve when he is below it. This is very useful since now it is possible to chance the distance between computed points.

As we chose a very high number of points during the interpolation we can control the distance by going through all of them and choosing the value after which we want to save a point in a straight and in a turn. A value of $0.1m$ for the straight line and $0.05m$ for the turn was chosen, as represented in Figure 12.

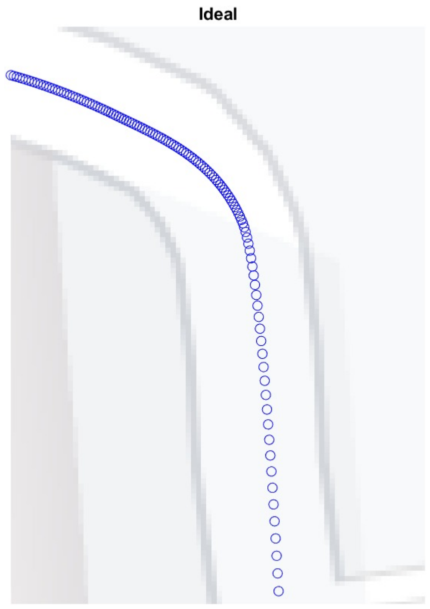


Figure 12 – *Difference in distance for computed points.*

Since a fixed value of time was chosen for the car to move between two given points, having the points closer during turns is a valuable addition, since it will make the car go slower and since those are the parts of the trajectory where we need to be more accurate.

Using the methods described previously it is possible to compute the array of positions of the car for the path selected, with the spacing we chose for straights and turns. The trajectory generator also computes the angles between two consecutive positions. The positions and angles computed will be used by the controller to compute the velocities and steering of the car to move from one point to the other.

4 Control

The control part includes the car model and the controller blocks. The objective of the controller is to determine the linear and angular velocities needed by the car to follow the path given by the guidance. The controller relies on the the difference between the real position of the car and the reference one described by the trajectory [1],[2],[3].

4.1 Car model

The first step, before we build the controller, is to design the car model in order to understand it's behaviour. We can consider the simple car model shown in Figure 13, with the car specifications represented in Table 1.

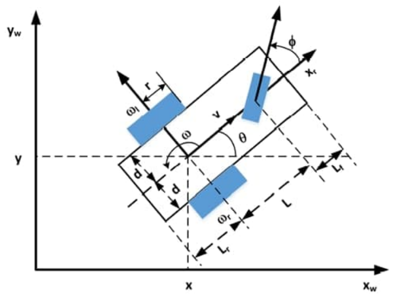


Figure 13 – *Kinematic of the car-model.*

L	2.2 m
Lr	0.566 m
Lf	0.566 m
d	0.64 m
r	0.256 m
Length	3.332 m
Width	1.508 m
Mass	810 kg

Table 1 – *Machine specification.*

The car moves in the (x,y) direction with a rotation with respect to its center equal to θ. It is important to note that we cannot consider that the car can freely move in space. To prevent unrealistic solutions, the steering wheel will turn with a rotation angle of Φ. In this way we are building a model of a car that will behave as a real car, preventing sideways movement.

First of all we had to define an initial condition for the car:

$$q_1 = (x_1,y_1,\theta_1) \tag{1}$$

that coincides with the first point of the ideal selected trajectory. In order to predict the next position of the car we have to solve the system, which originates from the differential kinematics of the car in Figure 13.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\Phi} \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 \\ sin(\theta) & 0 \\ \frac{tan(\Phi)}{L} & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v \\ \omega_s \end{bmatrix} \tag{2}$$

Solving this system means solving the following transition equations of the car:

$$\dot{x} = vcos(\theta) \tag{3}$$

$$\dot{y} = vsin(\theta) \tag{4}$$

$$\dot{\theta} = \frac{v}{L}tan(\Phi) \tag{5}$$

$$\dot{\Phi} = \omega_s \tag{6}$$

The last equation, 6, represents the fact that the steering wheel orientation cannot change instantaneously, but it will change orientation with a speed of ω_s in a finite time. Once the transition equations are determined, it is possible to discretize them in order to solve them:

$$\begin{aligned} q_{k+1} = (x_{k+1},y_{k+1},\theta_{k+1},\Phi_{k+1}) = \\ (x_{k+1},y_{k+1},\theta_{k+1}) + (\dot{x},\dot{y},\dot{\theta},\dot{\Phi})\Delta t \end{aligned} \tag{7}$$

For small values of Δt, that we set equal to $0.1s$.

Moreover, we have defined some limitations of the parameters; first of all the velocity was limited, as much as possible, to respect the speed limit present inside the campus. We also wanted to ensure that the movement of the car was smooth, so the steering angle is limited $|\Phi| < \frac{\pi}{4}$. The choice of this constrains allow to limit the steering wheel oscillations to better control the car.

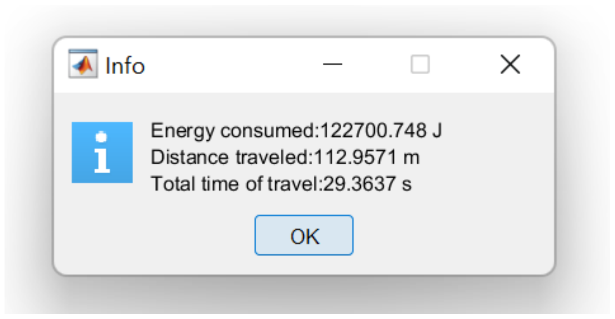
4.2 Energy consumption

An important feature to consider for an electric autonomous car is the energy budget. The activity of the car has a direct impact on energy expenditure, so for the whole path it is needed to keep track of the energy consumption and check that it is reasonable.

In order to monitor the energy consumed we need to consider the incremental energy required for each instant of time Δt. The energy spent at each cycle by the car is given by:

$$\Delta E = (M|\dot{v}(t)||v(t)| + P_0)\Delta t \tag{8}$$

where *v(t)* is the velocity of the car that is the output of the controller, $\dot{v}(t)$ is the acceleration with respect to the previous position, M is the mass of the car, present in Table 1, and P_0 is the constant cost of having the car in operation but not moving. At the end of the path a pop-up displays the amount of energy consumed, as in Figure 14.



Energy consumed:122700.748 J
Distance traveled:112.9571 m
Total time of travel:29.3637 s

Figure 14 – *Example of pop-up used to show the energy consumed.*

4.3 Controller

The objective of a controller is to adjust the real car trajectory, so it follows the ideal trajectory defined in 3.5. Following a trajectory means that the car has to reach a configuration q at time t. After reaching that configuration, the car has to be ready to reach the next one (q_{k+1},t_{k+1}) in the next time step.

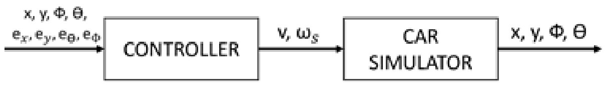


Figure 15 – *Schematic representation of the controller.*

As shown in Figure 15, the inputs of the controller are the states of the car (x,y,θ,Φ) and the world frame error w_e = (e_y,e_y,e_θ,e_Φ). The errors in the world frame w_e are computed as the difference between the car's position and the point on the trajectory that we have to reach:

$$w_e = \begin{bmatrix} x_{ref} - x \\ y_{ref} - y \\ \theta_{ref} - \theta \end{bmatrix} \tag{9}$$

The relationship between the world frame error w_e and the body error b_e is described by the system (10) derived from the Figure 16.

$$\begin{bmatrix} b_{e_x} \\ b_{e_y} \\ b_{e_\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} w_{e_x} \\ w_{e_y} \\ w_{e_\theta} \end{bmatrix} \tag{10}$$

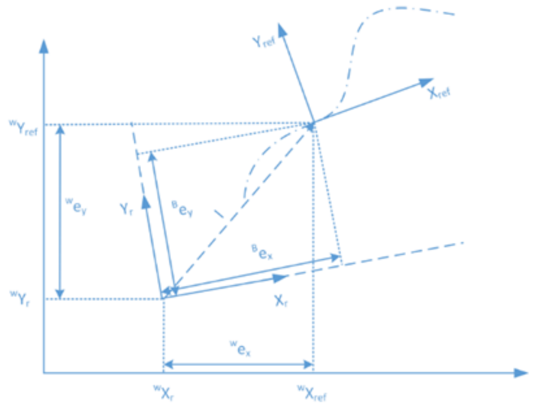


Figure 16 – *Relation between word frame and body frame.*

By solving the system it is possible to compute the body frame error, meaning the error between the actual position and the reference one, with respect to the car frame shown in 13. Once the body errors are determined the velocities v and ω_s can be computed as:

$$v = K_v \cdot b_{e_x} \tag{11}$$

$$\omega_s = K_s \cdot b_{e_\theta} + K_l \cdot b_{e_y} \tag{12}$$

where the parameter K_v, K_s and K_l have been tuned for this specific model.

K_v	0.5
K_s	100
K_l	1

Table 2 – *Gains values.*

These parameters give a weight to the body frame error of x, Θ an y, respectively. The output of the controller are the velocity v of the center of the car and the angular velocity ω_s.

4.4 Control results

For the example show in Figure 17, we can observe the difference between the ideal path and the one computed by the controller.

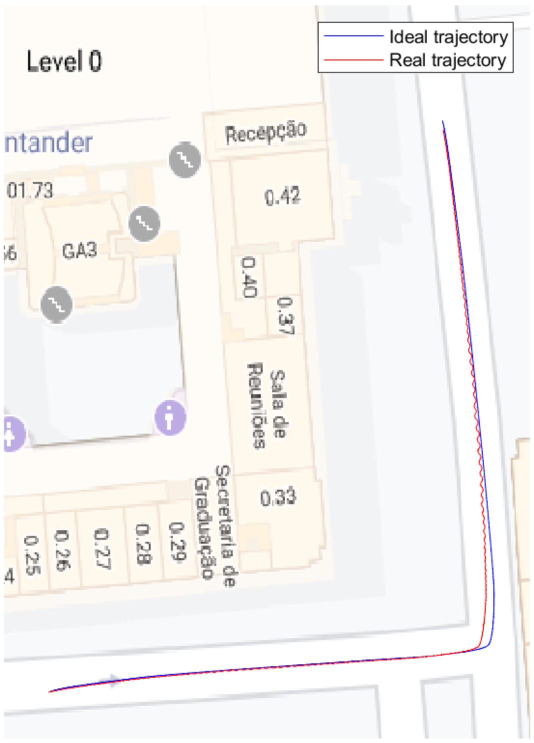


Figure 17 – *Comparison between real and ideal trajectories.*

Furthermore, the graph showing the variations in velocity is show in Figure 18. For this simple path it is easy to observe the behaviour of the car, how it accelerates in a straight line and slows down when reaching a turn.

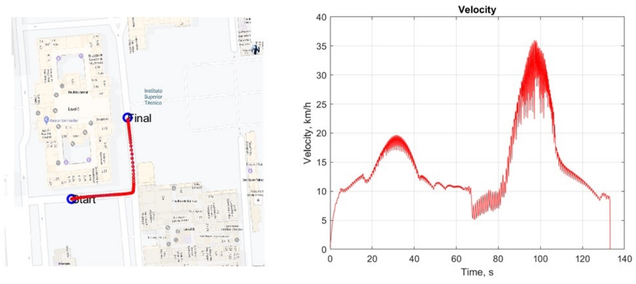


Figure 18 – *Example of path and velocity [km/h].*

5 Conclusion

The objective of this work was to develop an autonomous car able to compute the shortest trajectory between two given points inside the IST Alameda Campus, and compute a solution to move it from the start to finish positions. The program must also compute the energy spent during the movement of the car. To complete this task, a guidance and control sections were developed.

The guidance of the car was completed successfully, with it being able to compute the shortest path and giving the possibility of spacing the points in the trajectory with a chosen distance between them. Furthermore, the points between straight lines and turns can have different spacing's. One setback that was noticed during experimentation is the fact that if the starting and ending points are too close to each other, without any node between them, the program will not be able to compute the trajectory.

The controller is able to compute a trajectory that is reasonably close to the ideal one, keeping the car well inside the boundaries. The velocities are also reasonable, given that they aren't extremely high in a straight line and the car slows down when it has to do a turn, keeping it realistic.

Nonetheless, the control section was the one were the most problems were detected. The steering velocity during the path in unrealistically high and makes the steering wheel oscillate between positive and negative angles constantly. This may be caused by poor tuning of the control parameters, an over simplified controller or insufficient use of constraints. Other than that it was noticed that the car has problems following a trajectory from south to north (going up in the reference map) followed by a left turn.

Due to lack of time and human resources, these problems were not possible to solve. We will discuss in the following section future improvements that could be done to the work in order to fix it's problems and make it more realistic.

6 Future work and improvement

A future development will certainly be to incorporate the navigation block, present in Figure 19, which should be based on what the sensors detect. Sensor data should allow to predict carefully the coordinates of the car, and also allow the autonomous car to adapt to a changing environment (people crossing the road, traffic lights, etc.).

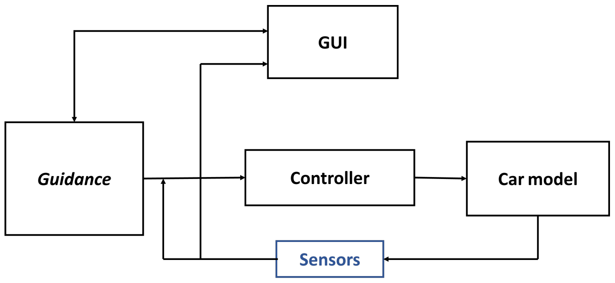


Figure 19 – *Complete block diagram.*

In order to deal with all of these variables the sensors that have to be placed on the car are:

- **GPS** that would provide a precise measure of the position of the car.
- **LIDAR** that stands for Light Detection and Ranging, a method for variable distance by targeting an object with a laser and measuring the time for the reflected light to return to the receiver[4].
- **Odometery** that is the use of motion sensors to determine the robot's change in position relative to some known position [5].
- **Camera** to capture the environment around the car.

Moreover, another important enhancement that should be done is related to the angular velocity and to the angle of the steering wheel. In fact, as shown in Figure 20, they present a completely unrealistic variation, even if a constrain on the angle of the steering wheel was considered.

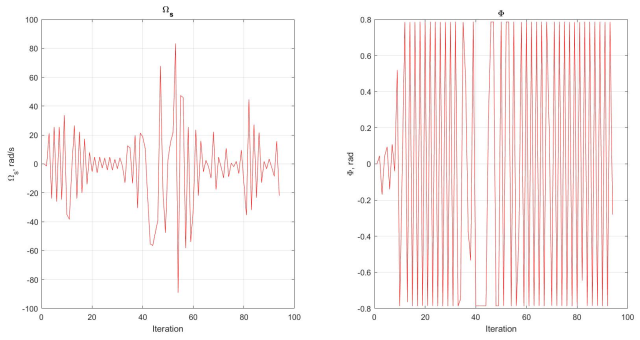


Figure 20 – *Example of Ω_s and Φ.*

Reducing those oscillations will lead to a more smooth real trajectory, even closer to a real driving scenario, and will drastically decrease the energy consumption of the autonomous car.

This problem could be caused by a not perfect tuning of the gains for the controller, reported in paragraph 4.3 inside Table 2. Another solution could be to implement a more complex and refined controller for this specific application.

In addition, while running some tests on the software developed we noticed some problems occurred when the car had to go from south to north, so basically from the bottom to the top of the map in Figure 6, followed by a left turn. In this specific situation the controller was not able to follow the ideal trajectory, leading the car to a completely wrong behaviour.

Another possible enhancement for this project could be the introduction of the energy consumption during the journey, displaying its value in the GUI, and decreasing it starting from an initial value, given to the software from the user. Doing this it would be possible to simulate the fuel of a real car, making the software closer to reality.

Table of Contents

List of Figures

References

[1] Daniel Vilela and Roberto Spinola Barbosa. Analytical models correlation for vehicle dynamic handling properties. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 2011.

[2] Ching-Hung Lee Ching-Cheng Ti-Chung, Kai-Tai Song. Tracking control of unicycle-modeled mobile robots using a saturation feedback controller. *IEEE Transactions on Control Systems Technology*, March 2001.

[3] J. S. Sequeira. Robotics, slide 6,7,8,9. *Robotics Course*, 2021-2022.

[4] National Oceanic and Atmospheric Administration (26 February 2021). What is a lidar. *US Department of Commerce.*, Retrieved 15 March 2021.

[5] Robo-rats locomotion: Odometry. `Robo-RatsLocomotion:Odometry`. Accessed: 2022-01-28.