Instituto Superior Técnico

Robótica

Laboratory 2

Autonomous Car

Alunos:

António Antunes - 90024

Francisco Velez - 90077

Pedro Salgueiro - 90164

Prof:

João Sequeira

Alberto Vale

9 de Junho de 2021

Contents

1 Introduction

The second laboratory of Robótica is about an autonomous car simulation inside a known environment. Problems like path planning, trajectory generation from a controller based on a real car model, event management: detecting and reacting, collision detection and energy monitoring were dealt with several strategies that will be further explained along this report. The outcome of this project is an app that allows the user to define the start, the end, via points and some events in a graphical interface to experiment the simulation of the autonomous car. In the other hand it provides also live data of that mission for a better and more transparent understanding of all the important moments of the car's run. It will be provided as an attachment a quick guide on how to operate with the app.

2 Car model

The considered car, as suggested, it was the one presented in (1). The model has 4 state variables: x and y as position coordinates, θ as an orientation angle and ϕ as the angle of the steering wheel. All of these are considered in the world frame. This car is assumed to be an electric car.

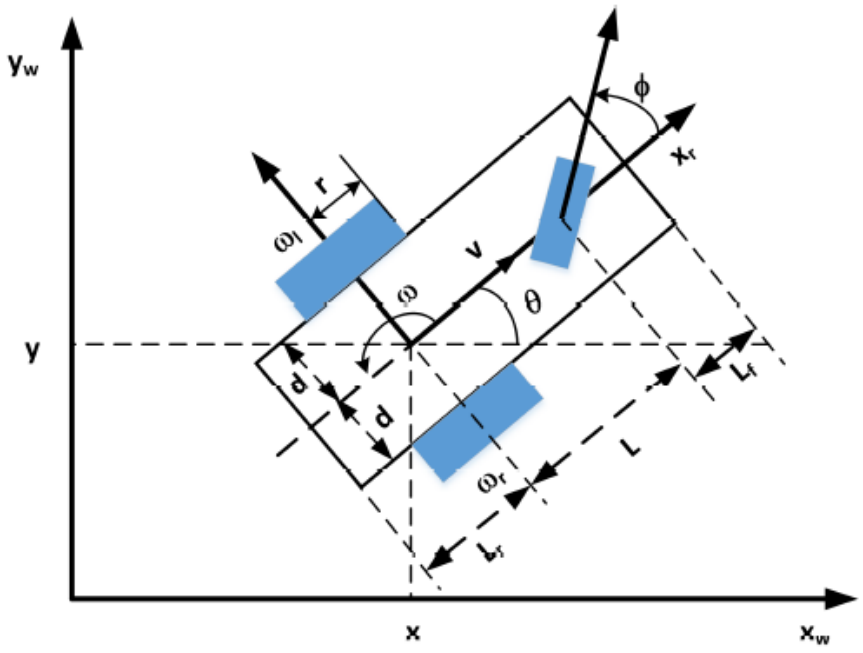


Figure 1: Kinematics of a car-like robot.

3 Environment

The chosen environment was, as recommended, the IST campus (2). We stick to it because it has narrow roads with many possibly challenging pathways to an autonomous car.

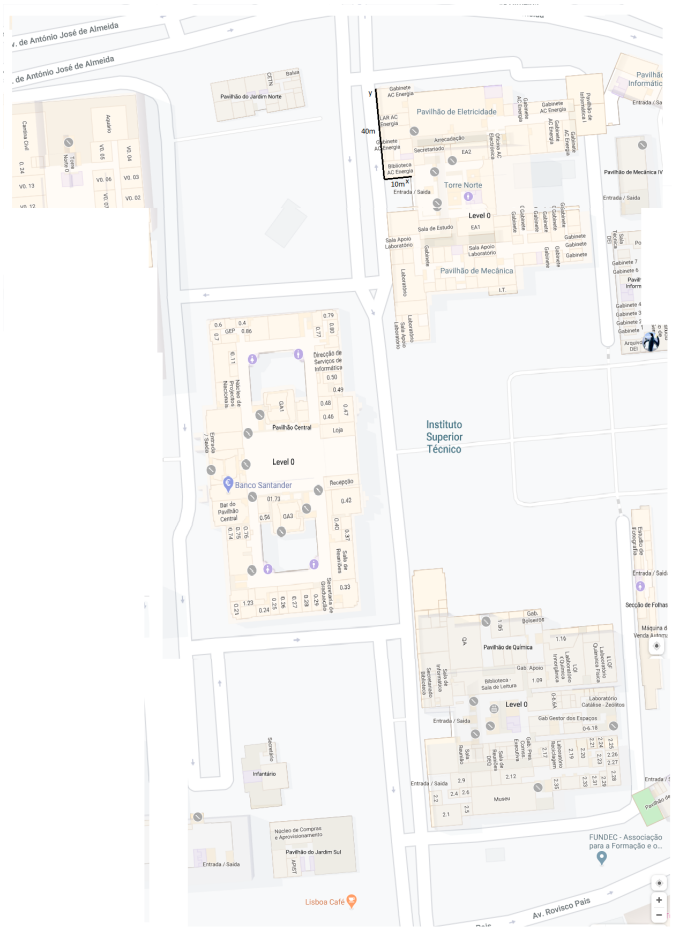


Figure 2: IST campus map.

The only information about this map is this image and a scale which is at the front of the north tower. It's important to point out that during all the process all computations regarding distances are done in pixels and later on converted to adequated to reality measurements in the graphical interface. The scale used was the same for x and y axis, 0.2 m/pixel.

4 Events

To set up a mission it is possible to define events. These events may be either static or dynamic.

4.1 Static Events

Stop traffic sign: This event creates a stop sign, in which the car must stop and then be able to continue its journey if it is planned to.

Figure 3: Stop sign.

Speed radar: This event creates a localization in which the car must pass by below a limited pre-specified speed by the user.

Figure 4: Speed radar.

Speed limited zone: This event creates a zone between two points in the road in which the speed of the car must be below a pre-specified limit by the user.

Figure 5: Speed limited zone.

4.2 Dynamic events

Crosswalk traffic light: This event is a traffic light fixed by the user with red and green, also user specified, time periods.

Figure 6: Crosswalk traffic light.

Pedestrians crossing the road: This event is a time and duration defined by the user in which at a certain point in the road, also defined by the user, pedestrians cross the road.

Figure 7: Pedestrians crossing the road.

5 Preparing the simulation

To execute a mission a preparation procedure must be followed, this is described in (8).

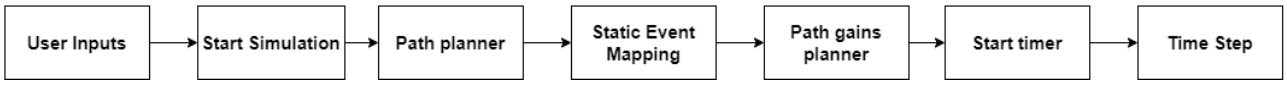


Figure 8: Preparation for simulation block diagram.

5.1 User Inputs

When the app is opened by the user a graphical interface pops up asking for an initial and final point for the mission, this points must be road points. It is also possible to set via points. Then there is an opportunity to select events to happen, some of them are static and others dynamic, all of them with pre-specified user defined parameters. As soon as the mission's constrains are all set the preparation is ready to begin.

5.2 Path planner

The first thing to do is to define the path that will feed the controller during the simulation. Here is the first challenge in terms of choosing a strategy to code what is a road and what is an obstacle. Which directions is it possible to go in each segment of the roads? The approach was a map based on (2) with a color based legend, as it is shown in (9). The red stands for upward direction, the green downwards, the blue to the left, the yellow to the right, white for all the directions and black for an obstacle.

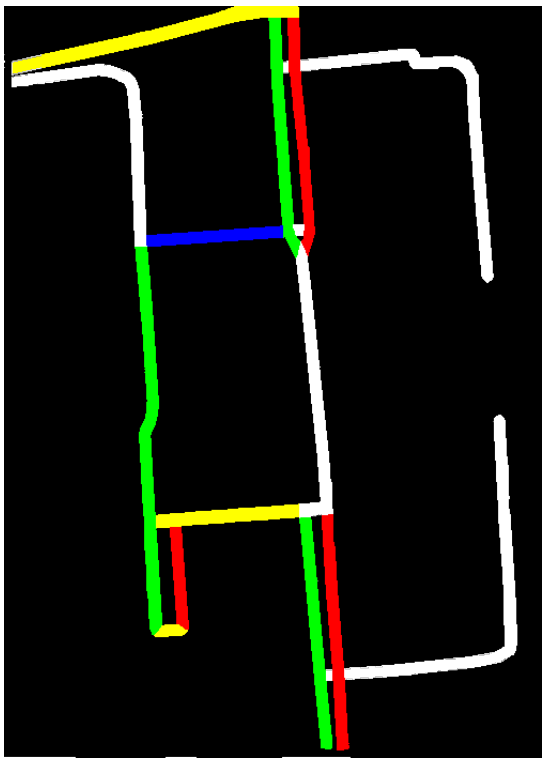


Figure 9: IST campus color map.

An important decision made was also to scale down the original image. This was done to reduce the amount of information dealt with into the *pathplanner.m* function. This colors made

it possible to create a graph in which the nodes (each road pixel) would only get to connect with its surroundings in a proper pre-defined by color direction. The weight of each connection is the absolute value of the inverse of the cubic distance (in pixel) to the closest obstacle in the arrival point.

A grid of the map is also generated in which the road pixels has the correspondent graph node number and the obstacles the value -1. From the moment the graph is created the next step is to assign to the start, end and via points each one's nodes id and then to use an algorithm to search the shortest path in the graph. In order to do that it was used the MATLAB function *shortestpath.m* in auto mode, therefore using the well known Dijkstra algorithm.

The final step to have a path defined by position and orientation based on a sequence of nodes is by cubic interpolation of those points using the function *csapi.m*.

5.3 Static Event Mapping

From now on the car will evaluate if its position is legal in terms of collision detection based on the grid with nodes and obstacles information. The events created by the user are mapped in this grid with a flag that carries an index to the list of events with all the important information. At this point only the static events are mapped, the others are dynamically mapped at each time step in a different function.

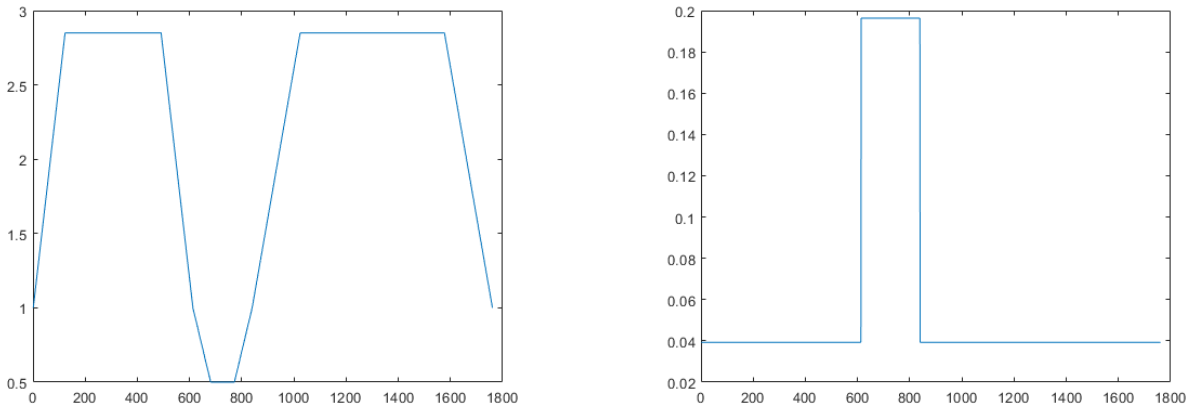
5.4 Path gains planner

To control the car to follow the desired path it is important to define the controller gains that will be later on explained in this report. We chose to define a dynamic smooth variation of gains K_v based on if the car is on a curve or if it is on a straight line. Along with this decisions a limit to the steering wheel angle was also defined for both situations.

To detect if the car is on a curve at some point in the path the correspondent orientation of the car was compared between a present point and a future point in the path array. If the absolute difference between the orientation is in $]0.298, 5[$ rad (values assigned by trial and error) a curve is detected. This approach has a problem that resonate into the controller operation in a bad way, it may assign several small curves into a big curve and counter curve situation. This problem was handled by joining any two curves if the distance between them is smaller than the sum of their own length.

As soon as the curve detection was solved the gains and limited angles were assigned to each

position of the path.



(a) (b)

Figure 10: (a) - Gains Kv, regarding linear velocity at each index of the path array. (b) - Angle lmitation ϕ of the steering wheel at each index of the path array.

The figure 10 is about a path with a straight line, then a curve and then another straight line until the end. The straight line gain is 2.85 and the limit angle is $\phi = \pi/100$ rad. On the other hand the curve gain is 0.5 and the limit angle is $\phi = \pi/16$ rad. It is also important to point out that the gains change state linearly in order to avoid abrupt oscillations of the gain K_v, and to allow the car to reduce speed as it enters a curve and to accelerate as it goes out of it.

5.5 Start timer

The simulation occurs in real time, a timer was defined with a time step of 0.1 seconds. At each time step the callback function *timeStep.m* run and process all the occurrences of each time step. This will be explained in detailed in the next section.

6 Time step loop

The callback function of the timer computes all the occurrences during each time step of 0.1 s and shares information through the graphical interface. The figure (11) presents its structure.

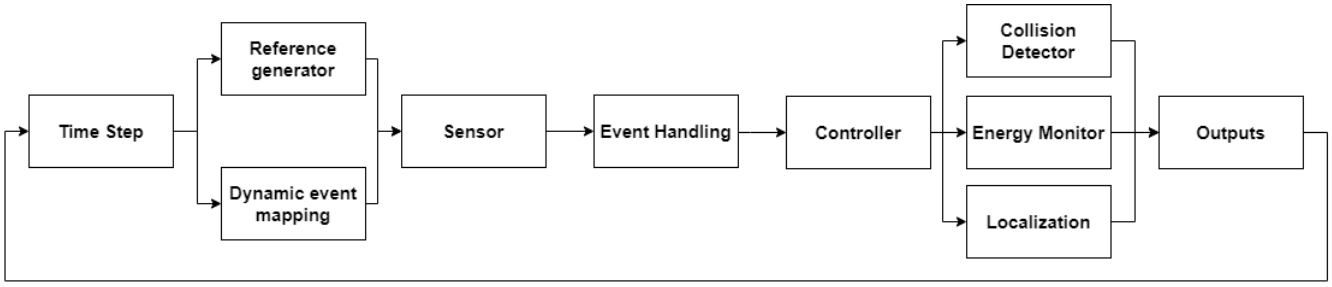


Figure 11: Time step block diagram.

6.1 Reference generator

The first problem to solve is to dynamically select a reference for position and orientation to feed the controller, which is way harder than just to follow a static reference. The strategy selected is to at first find which point in the path is the closest to the actual position of the robot in its trajectory. Then assign as references the position and orientation of a point in the path some indexes ahead (virtual car for trajectory tracking). After a lot of trial and error experiments the value selected was 50 indexes ahead in the path related arrays. After this procedure the references that feed the controller are defined in this time step, unless an event is later detected in which case this references must be adapted to face the adversity.

6.2 Dynamic event mapping

At each time step it is important to update the environment to enable the robot sensor to perform and detect possible events. Therefore the previously identified as dynamic events must be activated or deactivated accordingly to the pre-specified user's commands. In order to activate it a flag related to the event's index in the list of events is assigned in its position in the map. To deactivation a previously assigned flag is put back to zero in the map grid.

6.3 Sensor

To enable the robot of sensing its environment and autonomously react to events a sensor of proximity was designed. This sensor consists on the information of a 25 by 25 submatrix (an

approximate range of 12.5 m) around its current location in the grid that characterize the current situation of the environment. If events are found, the closest one in the direction of the trajectory is selected as the next event to fulfill. As a strategy to compare the direction of the trajectory and the direction from the car to the event, the signal of the dot product of both was used: if it is positive the direction is the same otherwise it is not.

6.4 Event Handling

If an event is detected by the sensor actions must be taken. Each event has its own procedure which is briefly explained below.

- **Stop traffic sign:** A new reference in the closest point in the path array to the position of the stop is assigned. The maximum speed of the car is reduced and it stops close to the reference. Then it continues its mission accelerating once again to its normal speed.
- **Speed radar:** A new speed limit is defined into the controller and the car passes by its location below that pre-defined limit.
- **Speed limited zone:** If the car passes by the start of a limited zone the maximum speed allowed into the controller drops to the new defined limit. If the car passes by the end of a limited zone the maximum speed is reset to its original value.
- **Crosswalk traffic light:** If the light of the traffic light was assigned as red by the dynamic event mapping the car behaves like it was explained for the stop sign except that this time it stops at a safety distance. While the red light is on the car stays stopped.
- **Pedestrians crossing the road:** The procedure is exactly the same as the crosswalk traffic light. The main difference is that this is a one time event and the traffic light is a cyclic event.

6.5 Controller

To control the autonomous car the equation (6.1) was used.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 \\ sin(\theta) & 0 \\ \frac{tan(\phi)}{L} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega_s \end{bmatrix} \tag{6.1}$$

Whenever (6.1) is computed white noise is added to all states, this noise has a maximum absolute value of 10% of the variation of the correspondent state variable in each time step. For example, if the variable x increase by 10 the noise added is in $[-1, 1]$, which would mean an uncertainty of 20 cm for a movement of 2 m. This model turned out to be relatively robust with respect to noise of this order of magnitude.

To generate the input of (6.1) a negative feedback based on a trajectory tracking reference was used. The tracking errors in the world frame are,

$$\begin{bmatrix} W_{e_x} \\ W_{e_y} \\ W_{e_\theta} \end{bmatrix} = \begin{bmatrix} x_{ref} - x \\ y_{ref} - y \\ \theta_{ref} - y \end{bmatrix} \tag{6.2}$$

To convert this errors into the car frame,

$$\begin{bmatrix} B_{e_x} \\ B_{e_y} \\ B_{e_\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} W_{e_x} \\ W_{e_y} \\ W_{e_\theta} \end{bmatrix} \tag{6.3}$$

This errors are multiplied with a matrix of gains which must be tweaked in order to get a better performance out of the controller. As it was said earlier the gain K_V, is related to the x axis in the car frame and therefore to the linear velocity, has a dynamic behaviour depending on whether the car is driving through a curve or a straight line. The gains related to the y and θ are fixed on $K_I = 100$ and $K_S = 1$ respectively. Therefore the input values for the equation (6.1) are computed by,

$$\begin{bmatrix} v \\ \omega_s \end{bmatrix} = \begin{bmatrix} K_V & 0 & 0 \\ 0 & K_I & K_S \end{bmatrix} \begin{bmatrix} B_{e_x} \\ B_{e_y} \\ B_{e_\theta} \end{bmatrix} \tag{6.4}$$

This model doesn't take into account mechanical limitations of a car, for example it is not sensitive to the previous time step linear velocity to decide its next linear velocity. To make the system more realistic some limiters were adopted. The first one is a maximum speed limiter, in any case the maximum speed at which the car can drive is 25 km/h, unless an event establish a lower speed limit. For simplicity it was chosen as maximum absolute acceleration 11.5 m/s^2, for both speed up and brake. It goes from 0 km/h to 25 km/h in approximately half a second. To prevent any liability that this model may have it was also decided that the car is not able to drive backwards, therefore it cannot make maneuvers in alleys. In terms of angular velocity it was limited

by an absolute value of $\frac{\pi}{4}$ rad/s.

If there is enough energy to accelerate as this model request in a certain time step the time derivative is computed in (6.1) and it is integrated to get the new values of each state. The new ϕ is limited as well accordingly to the pre-defined values that were defined in the path gains planner before the mission started.

If there is not enough energy the car just slide through the road to go as far as possible with the friction executing a counter force resulting in 50% of the pre-defined maximum acceleration. Which obviously will eventually make the car stop.

The energy spent at each time step is computed by,

$$\Delta E = (|F(t)|v(t)| + P_0)\Delta t = (M|\dot{v}(t)||v(t)| + P_0)\Delta t \tag{6.5}$$

It is important to mention that it was decided that braking would only spend P_0, therefore the acceleration present in the previous equation is related to positive acceleration. The value of P_0 is a constant value that represents the energy necessary to keep the electric motor working with constant speed. We chose $P_0 = 3$kW.

6.6 Collision Detector

Knowing the new position makes it possible to compute if there was a collision or not. Using the dimensions of the car, its position and orientation if any of the 4 corners of the car is in a position that is mapped as an obstacle or as a crosswalk the collision counter is incremented. If in consecutive time steps a collision is detected the algorithm only counts it as a single collision. This information feeds the graphical interface at each time step.

6.7 Energy Monitor and Localization

As the simulation goes by the energy and localization (position, orientation and speed) data flows from the callback function of the timer directly to the graphical interface, to present live the evolution of the car during its itinerary.

7 Simulation Test Example

As an example in figure 12 there is a test simulation that includes all the created events and via points, through curves and straight lines. The path has a traffic light, then at t = 6 s there are

pedestrians crossing the road for a duration of 6 seconds, there is a via point that makes the car go ahead instead of going to the left. There is a speed limited zone (v <= 18km/h) with a speed radar inside (v <= 15km/h). Then there is a challenging via point that forces the car to go all around the triangle on the map, finally there is a stop that forces the car to stop and then it carries on the remaining itinerary.

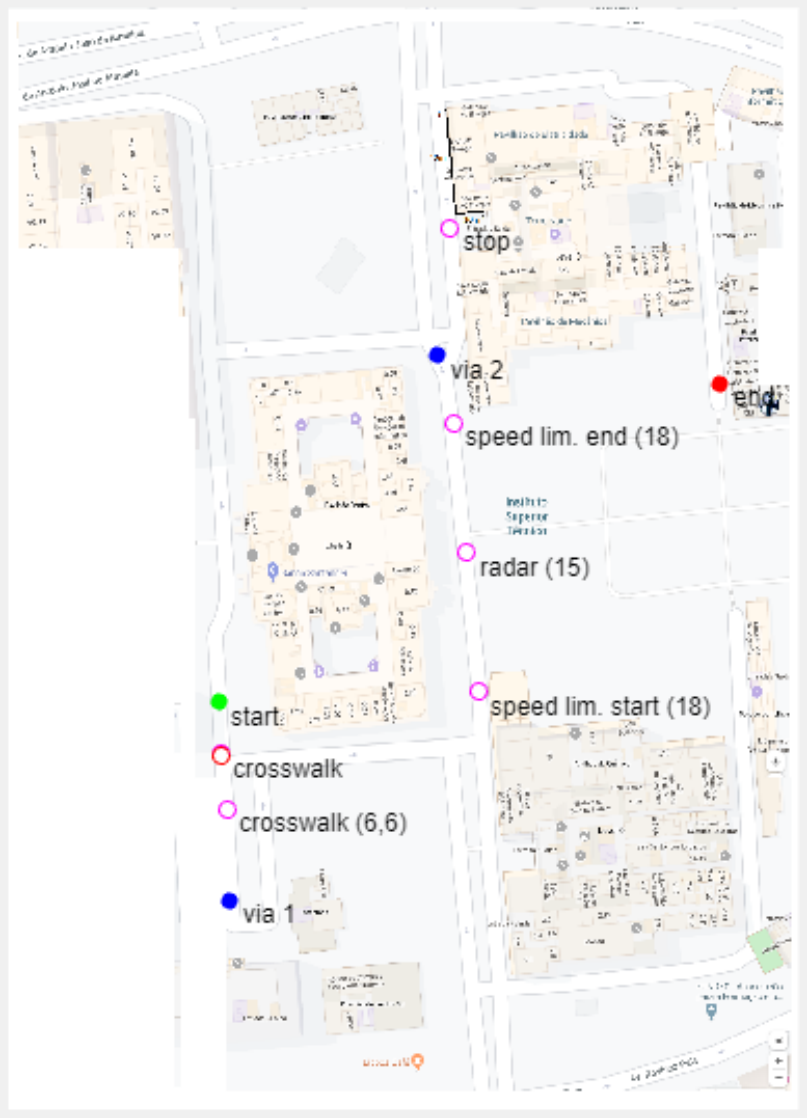


Figure 12: Example configuration for a test simulation.

The path assigned by the algorithm is present in figure 13. To see the path assigned to any mission uncomment the line 31 of the file *AutonomousCar_Lab2.m*.

Figure 13: Example configuration for a test simulation.

The trajectory described by the car by following the previous instructions is in figure 15. The mission took 181.3 seconds to be completed and there were no collisions registered.

Figure 14: Time and collisions monitor.

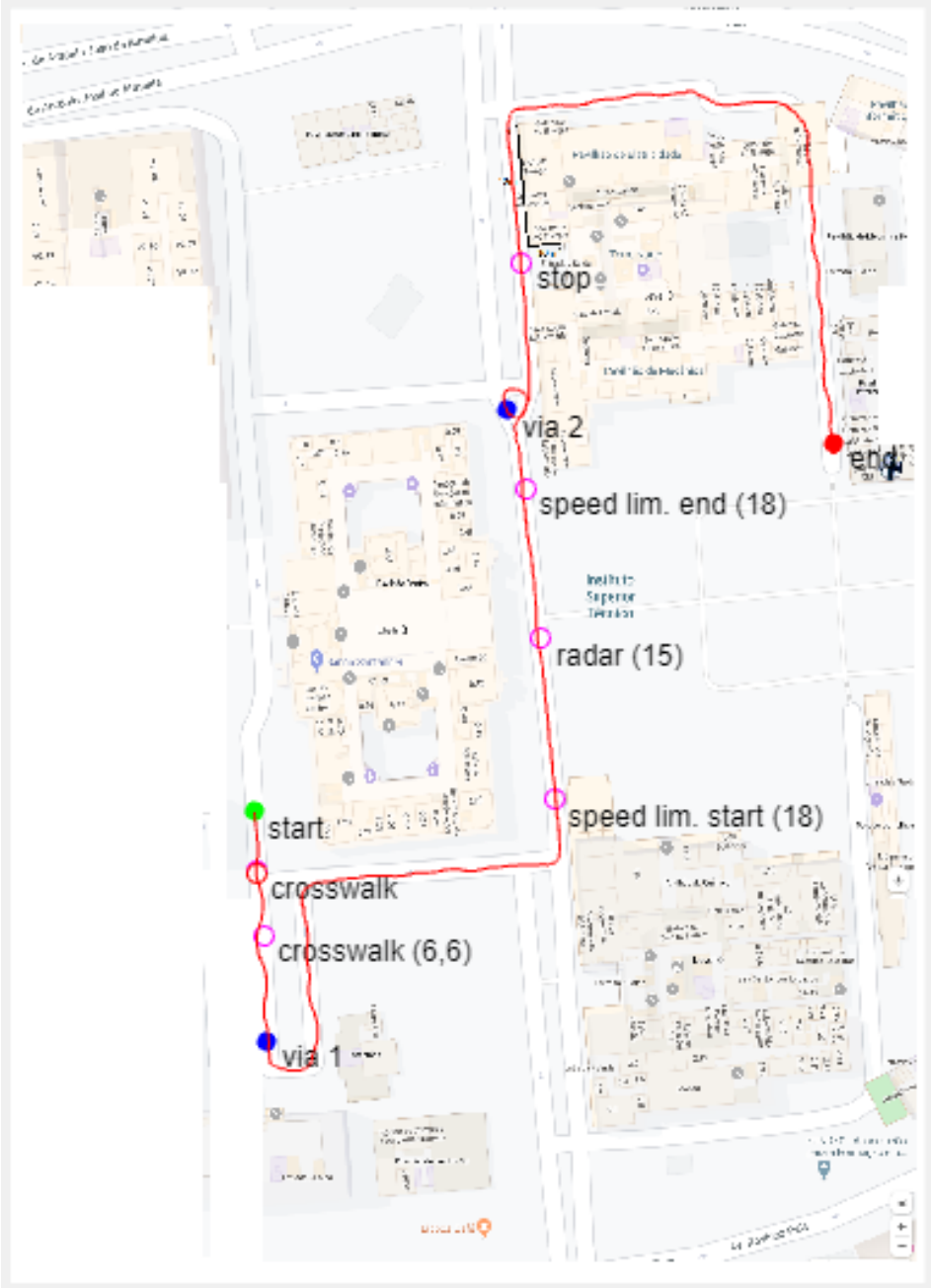


Figure 15: Trajectory of the car related to figure 12.

Through a first look at the figure it is possible to say that the trajectory is quite acceptable, however there are some moments with some oscillations. By looking at the plot of the linear velocity against time in seconds in figure 17, it is possible to see how the car managed its speed while handling with curves, straight lines or events, for example the four moments in which the velocity is 0 km/h that correspond to the two crossroads, the stop and the end of the mission. On the other hand it is also easy to see the behaviour of the car approximately between the 70s and

the 90s while it is driving through the speed limited zone and the fixed radar. By watching it live it is also easy to watch the car braking while entering a curve and accelerating while leaving it.

The evolution of the angular velocity in rad/s against time in seconds is described in figure **??**, it oscillates between three commands: stay still ($\omega_s = 0$ rad/s), turn right or turn left (same absolute value, but different sign). The turns happen frequently because the car is constantly adjusting to the trajectory, which is a palpable way to acknowledge the oscillations.

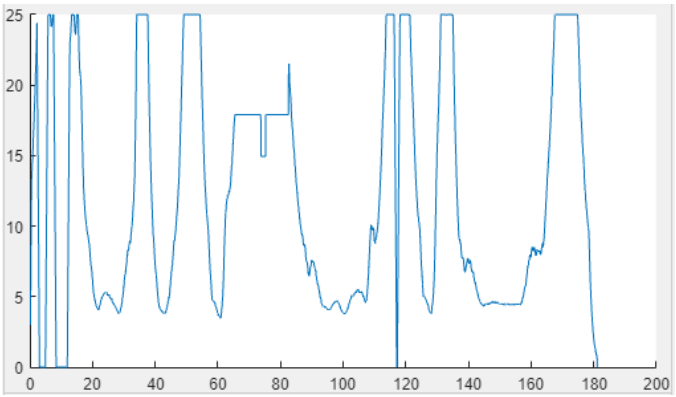


Figure 16: Linear velocity during the trajectory.

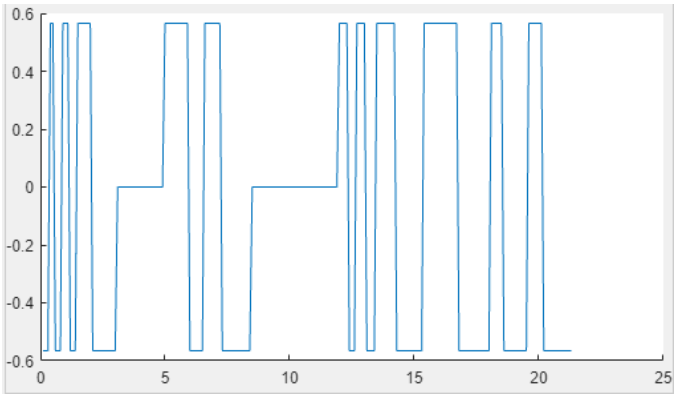


Figure 17: Angular velocity during the first 20 seconds of the trajectory.

There is in the graphical interface another live plot related to the energy spent in J against time in seconds, by watching it simultaneously with the linear velocity plot it is possible to see the energy being spent every time the linear velocity increases. The figure 18 presents all the trajectory energy management.

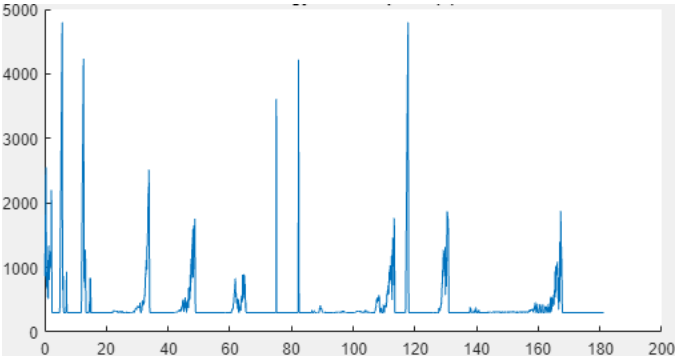


Figure 18: Energy spent during the trajectory.

It is visible that the peaks present in the linear velocity plot are actually correspondent in time to the peaks present in the energy plot. Furthermore it noticeable that the energy plot has an offset which is related to $P_0 = 3\text{kW}$, which correspond to a spending of 300J in a time step of 0.1 s.

The following table presents some statistics. The average speed end up being a little bit low, however it is acceptable because there are many events and curves that require decreasing of speed or even the car to stop. There are also information related to travelled distance and energy consumption, this information is merged into an estimate of the range of the car based on the available energy and the rate of spending until then. The ETA (estimated time to arrive) is a live indicator that displays a prediction, based on the data available about the journey until that moment, of the time left to finish the mission.

Statistics		
12.06	607.53	4.15
Average Speed (Km/h)	Travelled Distance (m)	Power Consumption (KW)
0	847.27	0.21
ETA (s)	Estimated Range (m)	Energy Consumption (KWh)

Figure 19: Statistics of the mission.

8 Conclusions

This simulation of an autonomous car is based on several approximations, for example: the whole environment being a plane, or choices based on group decisions like energy values, maximum acceleration values or even the fact that it runs through discrete time. These approaches together with the fact that the data referring to positions, orientations and speeds are not ideal values, with

consideration of noise, makes this simulation closer to reality. It is a good base, with some changes, for a possible implementation. In theoretical terms it has some useful choices specially in terms of path planning and following, or even in terms of handling events. The controller could be related to a better model, a more specific one to the actual car, which could be more robust in terms of dealing with noise. Due to that noise component added in the controller, close to objects passes of the car can be presented as collisions, that is a downside of not having a filter to mitigate the noise consequences. For example a Kalman filter could be applied to the noisy data in order to avoid those consequences. Another argument against the robustness of this simulation in order to implement it is the fact that, as it was said before, the car is not able to move backwards and therefore to make maneuvers. It is also important to mention that for some reason despite of using a timer the app it does not always run in real time, which is not acceptable for implementation in a real time physical system.

The final product of this assignment is an application which allows the user to experiment the simulation of an autonomous car that fulfills all the requested features, it travels from a starting point $(x, y)_{initial}$ to a final point $(x, y)_{final}$, it may pass by some pre-specified via points, and the most important: it does all that by following the shortest path inside the roads following the defined directions. It operates under velocity, acceleration and energy constraints and it handles all the desired events. It has two nuances for the E2 event, a radar and a speed limited zone. The simulation is also sensitive to collisions between the car and obstacles, both buildings and pedestrians in a crosswalk.

9 Quick Guide to the Autonomous Car Simulator App

9.1 App Overview

The app is composed of two main sections. A first section where the user can specify a mission for the AC (marked in figure 20) and a section section to display useful information and live data during the mission. Throughout this guide, the figure 20 will be referenced to illustrate specific parts of the app.

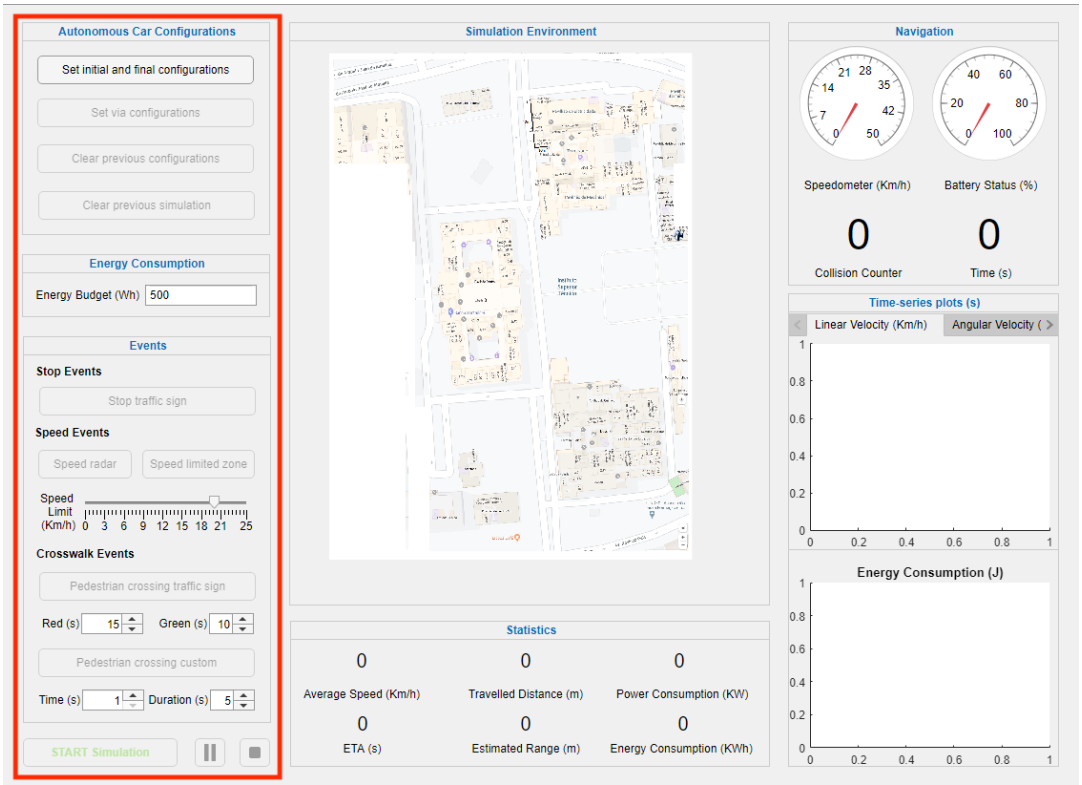


Figure 20: Autonomous Car Simulator App.

Additionally, the interface can be further divided into 7 main panels which group specific functionalities. These panels are: "Autonomous Car Configurations", "Energy Consumption", "Events", "Simulation Environment", "Statistics", "Navigation" and "Time-series Plots".

9.2 Preparing a Simulation

9.2.1 Setting initial and final configurations

click the **Set initial and final configurations** button. A figure will show up with the map (like in figure 21). Click once on the location of the initial configuration and click again to define

the final configuration.

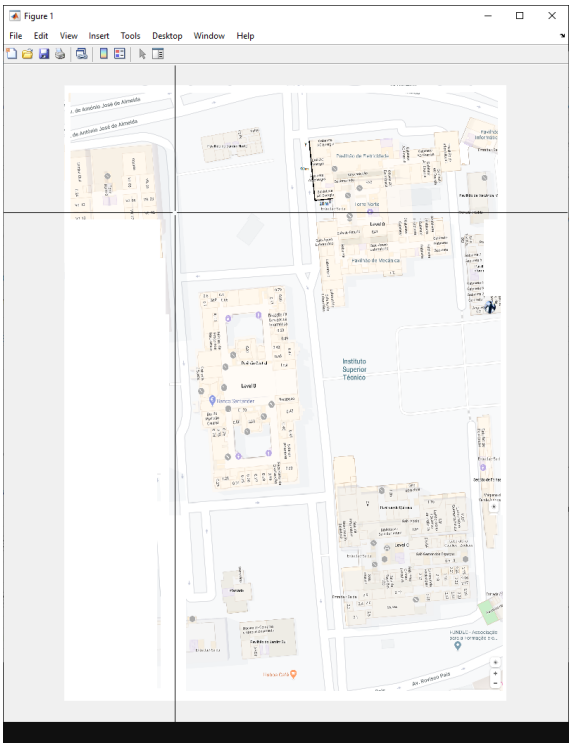


Figure 21: Selecting configurations on the map.

9.2.2 Setting via configurations

click the **Set via configurations** button. Click on the figure once for each via configuration to add. To end the process right click anywhere on the figure.

9.2.3 Changing the energy budget

In the energy consumption panel it is possible to change the AC's energy budget by changing the value in *Wh* in the **Energy Budget** text field.

9.2.4 Adding events

In the events panel there are a number of available events and configurations that can be set before starting the simulation.

- Adding a stop sign:

  Click the **Stop traffic sign** button to define the location of the stop sign. Click once on the popup figure to save that location.

- Adding a speed radar:

  Use the **Speed Limit** slider to set the maximum speed. Click the **Speed Radar** button to define the location of the speed radar. Click once on the popup figure to save that location.

- Adding a speed limited zone:

  Use the **Speed Limit** slider to set the maximum speed. Then, click the **Speed Limited Zone** button to define the beginning and end locations of the speed limited zone.

- Adding a crosswalk traffic light:

  Start by specifying the duration of the red and green lights using the appropriate fields labeled with **Red** and **Green**. Then, click the **Pedestrian crossing traffic sign** button to set the location of the the traffic light.

- Adding a crosswalk:

  Start by setting the time at which pedestrians start crossing and how much time they will take to reach the other side of the road using the fields labeled **Time** and **Duration**. Then, click the **Pedestrian crossing custom** button to set the location of the crosswalk.

9.3 Dealing with Multiple Simulations

9.3.1 Saving a configuration file

After preparing the simulation and/or alternatively to starting it, it is possible to save the configuration for later use. To do this click the **Save Current Configuration** button in the **File** menu (figure 22).

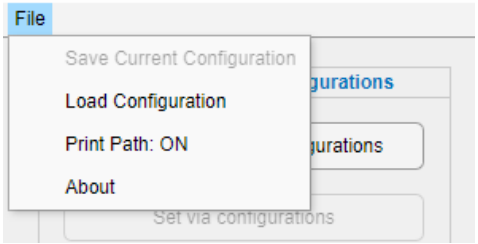


Figure 22: The **File** menu.

9.3.2 Loading a configuration file

Alternatively, load an already existing configuration file to the app and start simulating right away. Using the **Load Configuration** button in the **File** menu (figure 22) it is possible to select a previously saved configuration file containing all the information relative to the "Autonomous Car Configurations", "Energy Consumption" and "Events" panels. There are three example configurations already include with the program ready to be tested.

9.3.3 Clearing previous configurations

Click the **Clear previous configurations** button to start a new mission and clean the map on the "Simulation Environment" panel.

9.3.4 Clearing the previous simulation

If the current configuration is needed, it is possible to clear only the simulation information created during the previous simulation without clearing the mission configurations. To do this, click the **Clear previous simulation** button.

9.4 Running the Simulation

9.4.1 Before starting

Before starting the simulation check the "Simulation Environment" panel to make sure everything is ok. The figure 23 shows an example configuration. Also there is an option in the **File** menu to specify whether to print the path or not. This setting must be configured before starting the simulation.

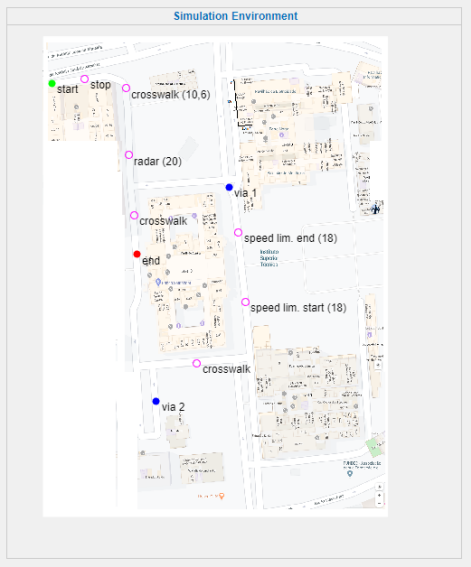
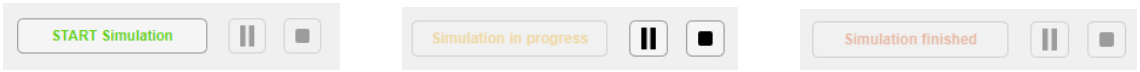


Figure 23: Example configuration.

In this representation, the filled circles represent the path configurations with green for start, red for end and blue for via configurations. The magenta circles with no-fill represent all the specified events.

9.4.2 Starting the simulation

After preparing all the configurations click the **START Simulation** button to begin the simulation (figure 24.a) The button will change to **Simulation in progress** during the simulation and to **Simulation finished** for a brief moment at the end before returning to its initial state.



(a) (b) (c)

Figure 24: Simulation state indicator: (a) - Before; (b) - During; (c) - After

Additionally, at any point in time during the simulation, the pause and stop buttons become available to pause the simulation at a specific part or to cancel it entirely.

9.4.3 Real-time simulation

The simulation is ran in real-time and there are a number of features included in this interface to better visualize and interpret the AC's movement and decisions.

- "Simulation Environment" panel

  In this panel an image of the map is shown along the real-time position of the AC and current status of all the static and dynamic events.

- "Navigation" panel

  Here, it is shown the current speed of the AC along with current battery state, number of collisions and simulation time.

- "Time-series Plots" panel

  In the "Time-series Plots" panel there are 4 plots. A tab selector on the top allows the user to change between the linear and angular velocities of the AC that are updated every time step with the instantaneous speed values for the AC along with the gain Kv and the bottom one works similarly but shows information relative to the instantaneous energy consumption.

- "Statistics" panel

  Finally, there is a statistics panel which includes real-time information on some data that is useful to keep track on. This panel displays the following information: average speed, ETA, travelled distance, estimated range, power consumption and energy consumption.

9.5 Robustness

Finally, the interface is designed to be handled by someone with little to no experience on the topic and first-time users will have no issues trying to use it. A lot of thought was given to the app design to make it as robust as possible and to not allow the user to make invalid requests to the program. An example of this would be to try to set initial and final configurations outside of the road space, or to start the simulation before setting the necessary parameters. All of these "safety features" prevent the program from crashing and make for an enjoyable experience when simulating the AC on the IST campus.