
Instituto Superior Técnico

MEEC

Robotics


# Autonomous Cars

Report


| **Students:** | **Number** |
|---|---|
| Carlos Silva | 81323 |
| Gonçalo Pereira | 81602 |
| Ivan Andrushka | 86291 |


June 5, 2020

# Contents

# 1 Introduction

In this work, a simulation for an autonomous car operating inside the IST campus is presented. The car follows a path defined by the user and acts in a reasonable manner when facing events (such as stop signs). The locations of the events are also defined by the user and can be changed when running new simulations.

# 2 Architecture

The architecture for the overall system is relatively simple and can be seen in Fig.1. The high level controller is responsible for generating paths between every pair of points selected by the user and dealing with events. The path points generated are then interpolated, in order to create a trajectory with a constant time-step between any consecutive points. The trajectory is fed to the low-level controller, which is responsible for following the trajectory.

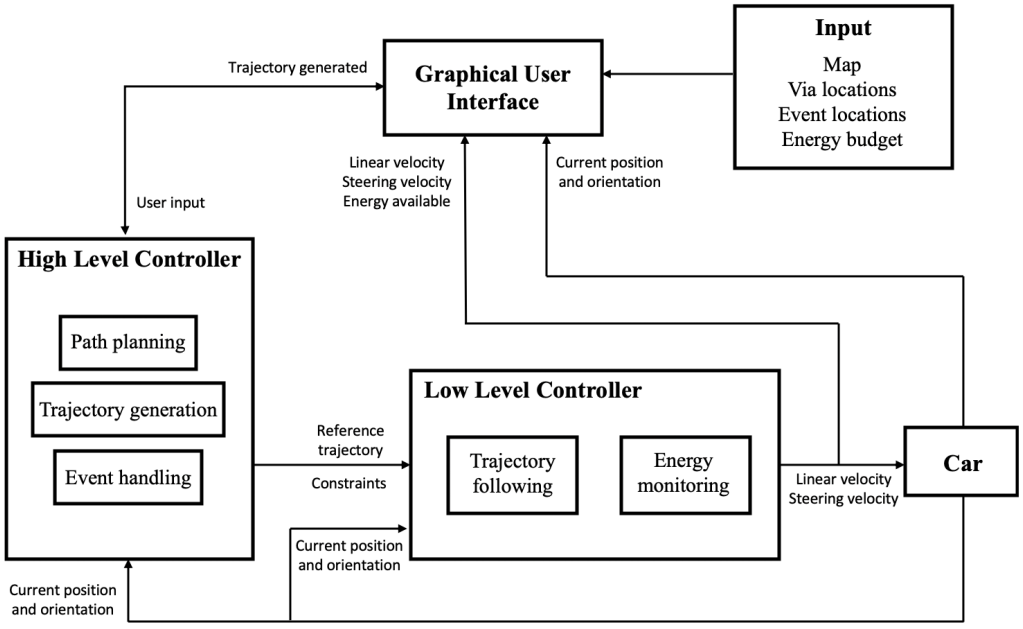


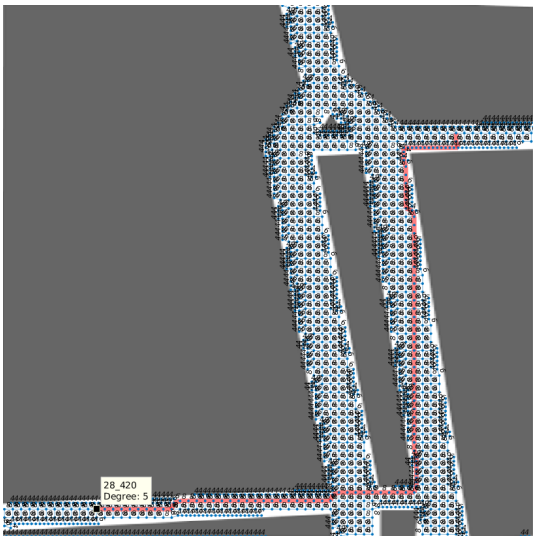

Figure 1: Architecture of the full system.

# 3 High level control

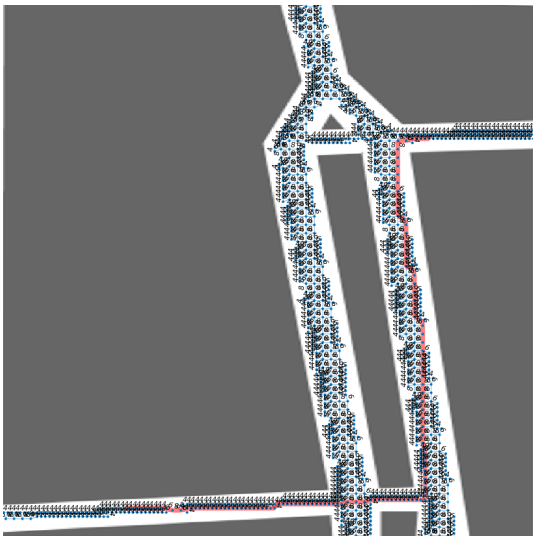
## 3.1 Path Planning and trajectory generation

To decompose the map, we first create a grayscale version of the map with the obstacles selected by the user. This is done by drawing filled polygons, which correspond to the obstacles, on a white image. Then, as a way to ensure that the car does not pass too close to the obstacles, the obstacles in the image are enlarged. Using the quadtree decomposition method on the resulting image we obtain an array of indexes and sizes for the resulting squares. We process this array creating a graph of adjacent free squares, using the middle of the square as the reference point. The connection weights are approximated based on the size of the connecting squares, adding up half the size of connecting squares.

Using this graph we map the reference points (beginning and end points included) to nodes picking the closest node to the given point. We find the shortest path between sequential reference points using Dijkstra's algorithm, generating arrays of connected nodes between given points. An example of the proposed solution can be seen in Figs. 2a and 2b.




(a) Quadtree decomposition and best path (red line) on a regular grayscale map.



(b) Quadtree decomposition and best path (red line) on a grayscale map with enlarged obstacles.

The trajectory associated with a given path is generated using a spline along the points of a down sampled version of the path. The down sampling the given path array serves the purpose of smoothing the trajectory.

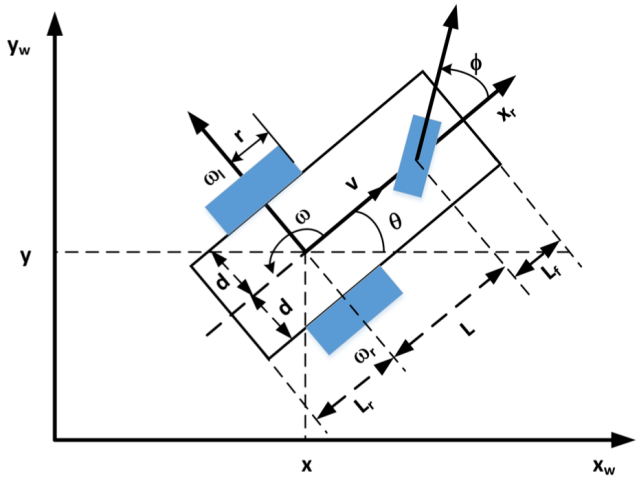
### 3.2 Event Handler

When the generated path passes through a neighbourhood of an event, a flag for said event is triggered. In the case of event E4 (pedestrian crossing road), the time of the event is also considered. The path is then divided in two segments. The first segment goes from the current position of the car to the position of the event. Upon reaching the end of the first segment the car executes some action depending on the event. Then, it continues its movement along the second segment, which goes from the position of the event to the end point of the initially selected path. The set of actions for each event is as follows:

- **E1** (stop sign) - The car stops at this event and stands still for 20 time steps.
- **E2** (speed limit sign) - After this event, the maximum velocity of the car is equal to the velocity specified by the user. The maximum velocity considered is 50, thus only values lower than this affect the velocity of the car.
- **E3** (pedestrian crossing sign) - Upon reaching this event the car slows down for some time, and then proceeds with its usual velocity.
- **E4** (pedestrian crossing the road) - Upon reaching this event, the car stands still for the amount of time specified by the user and then proceeds as usual.

## 4 Low level control

This module is responsible for movement of the car between any two arbitrary configurations $(x, y, \theta)_{initial}$ and $(x, y, \theta)_{final}$ inside the IST campus. The car model used to described its movement was the one studied in theoretical classes. The kinematics of the car as well as its dimensions is specified in Fig.3.




| Specifications | |
|---|---|
| $L$ | 2.2m |
| $L_r$ | 0.566m |
| $L_f$ | 0.566m |
| $d$ | 0.64m |
| $r$ | 0.256m |
| Length | 3.332m |
| Width | 1.508m |
| Mass | 810kg |

Figure 3: Car kinematics and dimensions.

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), t) = \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \\ \dot{\phi}(t) \end{bmatrix} = \begin{bmatrix} \cos(\theta(t)) & 0 \\ \sin(\theta(t)) & 0 \\ \tan(\phi(t))/L & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v(t) \\ \omega_s(t) \end{bmatrix} \tag{1}$$

with $\mathbf{x} = \begin{bmatrix} x(t) & y(t) & \theta(t) & \phi(t) \end{bmatrix}^T$

For simulation purposes, we used the Euler numerical integration method in order to get a first order approximation of the car model.

$$\mathbf{x}_{n+1} = \mathbf{x}_n + hf(\mathbf{x}_n, t_n) \tag{2}$$

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \\ \theta_{n+1} \\ \phi_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \\ \theta_n \\ \phi_n \end{bmatrix} + h \begin{bmatrix} \cos(\theta_n) & 0 \\ \sin(\theta_n) & 0 \\ \tan(\phi_n)/L & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_n \\ \omega_{s_n} \end{bmatrix} \tag{3}$$

with $t_{n+1} = t_n + h$ and the step size, $h = 0.01$.

The next section will address the various steps involved in the design of the controller for the movement of the car, along the trajectory fed by the high level controller. The low level controller will return as output the pose of the car at the end of the movement, the controls (linear velocity and steering velocity) at each instant, a flag signal that warns if the car is going backwards ($|\theta_n| \geq \pi/2$) as well as some important information about the performance of the car (energy spent, distance covered and time elapsed).

## 4.1 Control strategy

A generalized architecture (block diagram) of this low level module can be found in Fig.4. Each block will be characterized in the following sections.

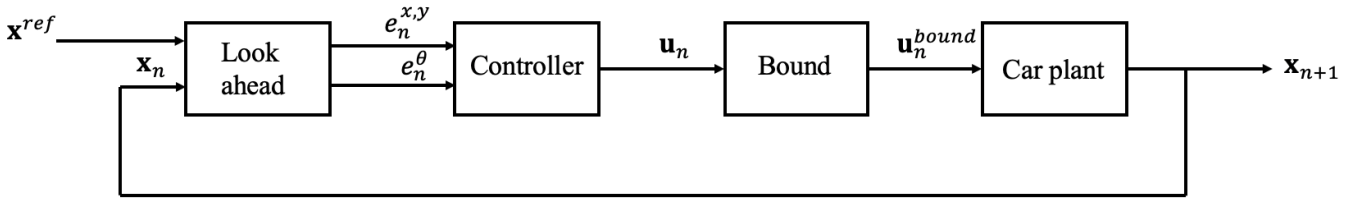



Figure 4: Block diagram of the closed loop low level controller.

### 4.1.1 Look ahead

Let $\mathbf{x}^{ref}$ be the reference trajectory generated by the high level controller module discussed in section ???. The initial position and orientation of the car is assumed to be close to the first point of the trajectory. Since we're solving a tracking problem, we wish that following conditions are true when for the system.

$$\lim_{t_n \to \infty} ||\mathbf{x}^{ref} - \mathbf{x}_n|| = 0 \tag{4}$$

$$\lim_{t_n \to \infty} ||\mathbf{x}_n^{ref} - \mathbf{x}_n|| = 0 \tag{5}$$

Based on the current state (position and orientation) of the car, we compute the closest point to the reference trajectory.

$$\begin{bmatrix} \hat{x}_n^{ref} \\ \hat{y}_n^{ref} \end{bmatrix} = \min_{x,y \in \mathbf{x}^{ref}} \left\| \begin{bmatrix} x - x_n \\ y - y_n \end{bmatrix} \right\| \tag{6}$$

Then, from that point we look for the point that is $K$ steps ahead and calculate the tracking errors, $e^{x,y}$ and $e^{\theta}$.

$$e_n^{x,y} = \left\| \begin{bmatrix} \hat{x}_{n+K}^{ref} - x_n \\ \hat{y}_{n+K}^{ref} - y_n \end{bmatrix} \right\| \tag{7}$$

$$e_n^{\theta} = \hat{\theta}_{n+K}^{ref} - \theta_n \tag{8}$$

The orientation error is also reduced to the smallest interval, i.e., $e_n^{\theta} \in ]-\pi, \pi[$.

### 4.1.2 Controller

For the car control, the group used the following non-linear control law.

$$\mathbf{u}_n = \begin{bmatrix} 1 - \tanh\left(K_v e_n^{x,y}\right) \\ -K_l \operatorname{sgn}(\alpha) e_n^{x,y} + K_s e_n^{\theta} \end{bmatrix} = \begin{bmatrix} v_n \\ \omega_{s_n} \end{bmatrix} \tag{9}$$

where $K_v, K_l, K_s > 0$ are tuning constants for the controller. Moreover, $\alpha$ is the third component of the following cross-product.

$$\begin{bmatrix} \hat{x}_n^{ref} - x_n \\ \hat{y}_n^{ref} - y_n \\ 0 \end{bmatrix} \times \begin{bmatrix} \cos\left(\hat{\theta}_{n+K}^{ref}\right) \\ \sin\left(\hat{\theta}_{n+K}^{ref}\right) \\ 0 \end{bmatrix} \tag{10}$$

The sign of the angle $\alpha$ allows us to know in which side, relative to the reference trajectory, the car is. If $\alpha > 0$ the car is on the left side and on the right side if otherwise.

### 4.1.3 Bound

Even though the controller drives the position and orientation errors asymptotically to zero, there are still a few problems to solve. The generated linear velocity, $v_n$, and steering velocity, $\omega_{s_n}$, can be "too much" for the real system to handle, for example, in terms of the energy available to the car in that instant. Therefore, the car plant inputs must be bounded to more suitable values in order for it to behave like a real car would.

The constraints used to solve this problem were the following

- $|v_n| < 50$km/h - Speed limit inside the IST campus.
- $|\omega_{s_n}| < \pi/8$ - Maximum steering velocity.
- $E_n = \sum_{i=0}^{n} \Delta E_i < E^{budget}$ - Energy constraint which forces the car to move with limited energy. The tracking of available energy is done within this block (the implementation of this feature is described later on).
- $|\theta_n| < \pi/2$ - The orientation of the car must not surpass $\pi/2$ rad which would mean that it's going backwards.
- $|\phi_n| < \pi/4$ - Limitation of the steering angle.
- $|v_n| < v_{max}$ - This condition is established based on the energy available to the car. The value of $v_{max}$ is explained in Section 4.1.4.

### 4.1.4 Energy consumption

The energy consumption of the car is monitored along its trajectory. The tracking of the energy spent by the car is done in this module according to the following steps:

- Keep track of the energy spent up to the current instant, $E_n = \sum_{i=0}^{n} \Delta E_i$. In each movement $\Delta E_i$ is calculate according to the following expression:

$$\Delta E_i = (M \frac{d}{dt} v_i + P_0) v_i h \tag{11}$$

where $P_0$ is a constant value.

- Compute the availabe energy for the remaining of the trajectory $\Delta E_n^{budget} = E^{budget} - E_n$.
- Assume that the available energy per step is $\Delta E_n^{budget}/N_{steps\ remaining}$.
- Set the maximal linear velocity to the maximum value of the solution of the equation: $(M\dot{v} + P_0 = \Delta E_n^{budget}/(\Delta t N_{steps\ remaining})$. The solution of this equation (shown below) is used as a conservatibe estimate for the maximum velocity in order to allow the car to reach the end of the trajectory.

$$v_{max} = \frac{\Delta E_n^{budget}}{P_0 \Delta t N_{steps\ remaining}} \tag{12}$$

## 5 Graphical user interface

The layout for the user interface can be seen in Fig.5.

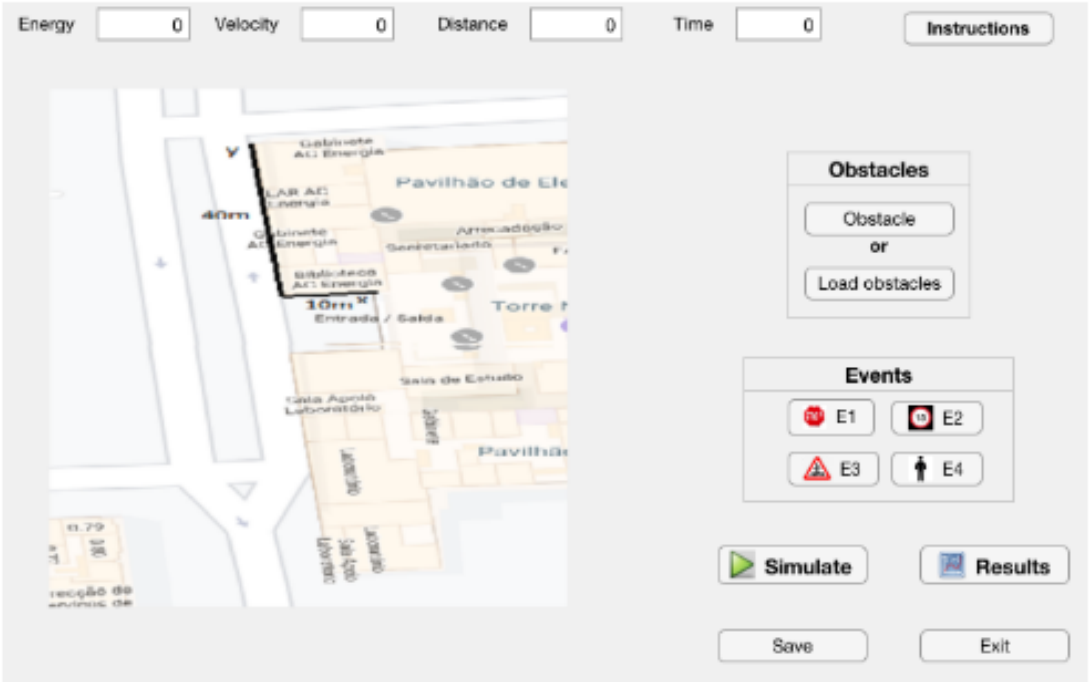



Figure 5: Layout of the graphical user interface

The interactive buttons are

- **Instructions** - opens a .txt file in the built-in MATLAB text editor with a summarized list of instructions.
- **Obstacle** - opens a new window where you can select an obstacle. This process has to be repeated for each obstacle.
- **Load obstacle** - loads the set of obstacles that has been saved in previous runs of the application.
- **Events** - click on each event E1, E2, E3 and E4 to set their locations on the map.

- **Simulate** - begins the simulation.
- **Results** - shows the plots of the control variables and the trajectory look ahead error.
- **Save** - saves the previous object configuration. Also saves the graph generated for said configuration if you have simulated the system at least once.
- **Exit** - safely exits the app.

The non-interactive display boxes are

- **Energy** - displays the amount of energy available at each step.
- **Velocity** - displays the current velocity of the car. The velocity control varies between 0 and 1 but the display, for presentation purposes, is scaled to be between 0 and 50.
- **Distance** - displays the distance travelled between each segment of the path.
- **Time** - displays the simulation time

# 6 Experiments

The expected outcome is that the car must move smoothly along the reference trajectory, react accordingly to the different events in its path and the controllers outputs must be reasonable compared to a real life situation. Figure 6 shows the simulation window with the chosen path (and a STOP event along the way). The controls along this trajectory are plotted in figure 7.

Due to an implementation mistake, the controller outputs correspond only to the last section of the trajectory (yellow line) in figure 6. Although the car follows the reference trajectory accurately, its movement is not as smooth as we would have hoped. It's visible that the steering velocity $\omega_{s_n}$ has large amplitudes, causing the car be constantly turning its wheel. The linear velocity, however, seems to be a bit more smooth. These results show that the option to use a non-linear controller was not the best since we would have to keep fine tuning the controller gains, as well as the look ahead steps.

In regards to the generated path, we can see that it tends to pass close to the obstacles. This could be tuned by enlarging the obstacles more, but some narrow paths could be entirely blocked if the obstacles are enlarged too much. Thus, the proposed solution might not be adequate for smaller images, such as the one used in this map. On the other hand, we can see that it performs adequately on zoomed images such as the ones in Figs. 2a and 2b.

# 7 Conclusions

In this project, a software that simulates an autonomous car was developed. The path planning module was based on finding the shortest path on a visibility graph obtained through quadtree decomposition. This method performed adequately in most situations, but required some more fine tuning, as in some cases the optimal path would pass very close to obstacles. The trajectory following control was implemented by means of a nonlinear Lyapunov controller. While the car, in general, followed the trajectory, the control was not very smooth.

In conclusion, while the car performed as expected, the trajectory following controller and path planning modules could be further fine tuned to provide better results.

Figure 6: Simulation window.

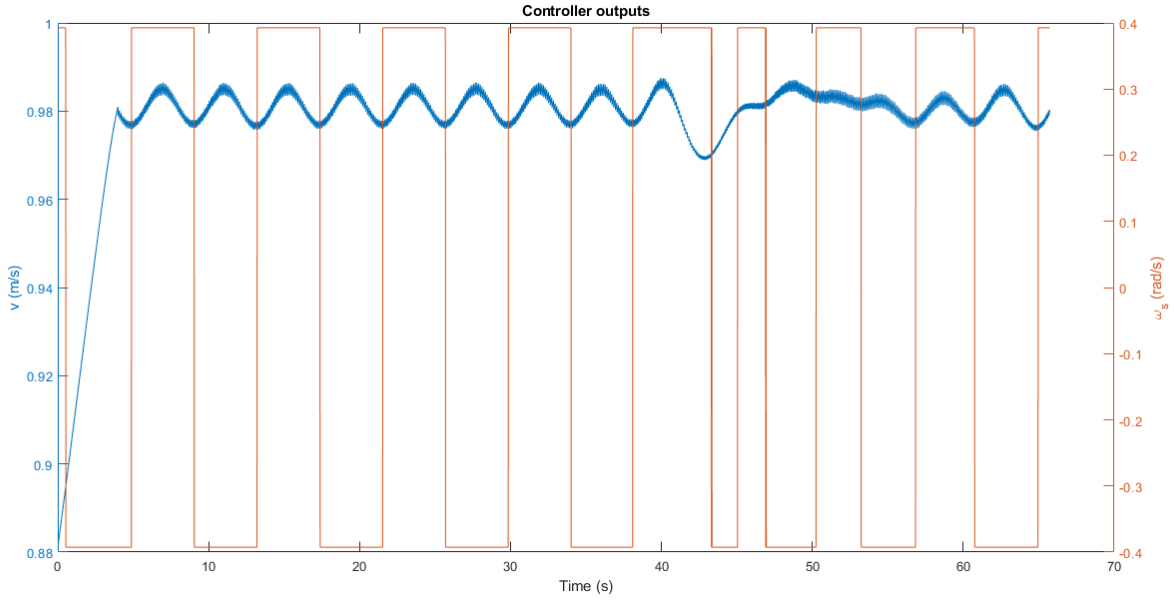



Figure 7: Controller outputs. Linear velocity - blue. Steering velocity - orange.