Electrical GNC Autonomous Car

Guidance Team
Authors {IST id}: ALMEIDA. João {90119}, ROSA. Daniel {90041} and VIEGAS. Guilherme {90090}
Email: { *joao.marques.almeida; guilherme.viegas; daniel.garcia.domingos.rosa* }@*tecnico.ulisboa.pt*

Navigation Team
Authors {IST id}: BRANCO. Catarina {90036}, CORDEIRO. Rafael {90171} and MARTINS. Ana {90011}
Email: { *catarina.m.branco; rafael.andre.alves; ana.r.martins* }@*tecnico.ulisboa.pt*

Control Team
Authors {IST id}: ORNELAS. Marco {90132} and SALDANHA. Paulo {90156}
Email: { *marcoornelas; paulo.saldanha* }@*tecnico.ulisboa.pt*

June 30, 2021

***Abstract*—The real-time simulation of an Autonomous Car performed, is formed by 3 main components: Guidance, Navigation and Control. The software has a GUI where many real-life events can be drawn (e.g. speed limit zones or random pedestrian zones) and a route based on checkpoints which are used in the dynamical Dijkstra bearing in mind that the direct cost of moving to a specific position differs based on the previous one. Finally the smoothing technique bspline in transitions is performed in order to make a continuous path more polished with a fixed sample rate. In addition the sensor fusion used in the navigation turns all simulation more real by making the autonomous vehicle aware of traffic lights, signs or potential dangers caused by pedestrians or localisation errors, that are simulated with gaussian noise in the EKF predictions based on the GPS information and the car dynamic model. It is also considered possible GPS anomalies such as breakup zones or a kidnapped situation. The available energy of the car is crucial for the velocity planning, during the path-following based on a virtual robot (avatar). The controller also takes into account the future route in order to smooth the linear and angular velocity transitions of the vehicle, which also simulates brakes for different road conditions (e.g. normal and wet tar).**

***Keywords*—** GUI circuit generation, Path-Planning, Dynamical Dijkstra, Path-Smoothing, Bspline; Sensor Fusion, EKF, Energy, Car Dynamics Model; Path-Following, Linear and Angular Velocity, Brake Simulation, Virtual Robot;

II. Introduction

AUTONOMOUS cars are a developing technology which already proves to be the next evolution in personal transportation, but for this, many things have to be accounted for. From knowing the environment where the car is driving to deciding the route to take, from deciding what controls should the car take to move in that path and even knowing where the car currently is plus the need to account for errors and noise in both sensors and actuators are just a few of the viewing angles one should look in order to build a realistic autonomous car.

Because building an autonomous car is not an easy task, the problem is usually and will be divided in this project into three parts, namely *Guidance*, *Navigation* and *Control*, usually known by its acronym *GNC*. The Guidance group will focus firstly on the framing of the maps used, from enabling a user to define the driveable roads and path points, to enable him to define obstacles, crosswalks and other important road marks. Secondly, this group is responsible for planning the path that will be given initially to the other groups, where the main worries are to plan the shortest path passing as close as possible to the given points, starting and ending with the given orientations. Finally, this group will also take care of smoothing the path to pass to the other two groups. The Navigation group main focus is in improving the car position and orientation estimate, using a set of sensors such as a GPS and an orientation sensor together with the application of an Extended Kalman Filter to reduce the error associated with the normal Control estimate. This group has the second job of detecting road marks such as traffic lights, and crosswalks using a camera and also detect unpredictable real-time behaviours such as pedestrians crossing a street, using a LIDAR sensor, where all of this information is passed to the control group to be handled. Finally, the Control group, taking into account the information received from both the Guidance and the Navigation group will take care of applying the dynamic equations of the robot to generate two control signals, the wheel turning speed and the linear velocity, so that the autonomous car moves according to the expected route. This group is also responsible for regulating the car's behavior, with respect to its energy and motion equations. One important note to take is the large communication that has to exist between the three groups to deliver a well designed autonomous car. One part cannot work without the others, and a successful work from one part is not only good for itself but also for the whole project.

This report is divided into three parts, each addressing one of the previously mentioned categories of a *GNC* system, where each group will first address the theory behind its work and then provide experimental results that show not only the overall performance of its part but also some specifics of importance to this report. In Chapter III a tutorial on how to run the project is provided. Chapters IV, V and VI will respectively take care of the *Guidance*, *Navigation* and *Control* parts of this project. Chapter VII shows the overall results obtained for two scenarios, both tested in the oral discussion. Chapter VIII will end this report with a reflection of the overall accomplishments and possible improvements.

III. Ease of use

TO run the software one can simply run the *main.m* script and a flow of pop-ups will appear depending on the user input. The first pop-

up asks the user if he wants to use a map image already defined by the group or if he wants to input a new image. For inputting a new image the user needs to enter the path to the image with the crucial step of having the image stored in a folder with the same name (except the extension) and this folder saved inside the folder 'maps/', for example for the image 'ist.png', it should be saved has '../maps/ist/ist.png'. From there the user specifies if it is a real image (taken from google maps for example) or just a sketch (a race track drawing). If a real image is chosen, then the user is asked to input two points in the map and the respective world coordinates to make the correspondent ratio between meters to pixels. In the sketch, a simple value for this ratio is asked.

Then the user is asked to define the roads for the map, where he simply needs to draw regions/polygons in the map to define zones for the car to drive. These regions can be drawn over one another, for example when drawing a ring like road, the user should draw a moon shape region and then the other moon shape region with the extremes over the extremes of the first.

The roads will appear in grey and after this, the user can define some road marks. From top to bottom, the user can choose to define 1) crosswalks, drawn in white, where he simply needs to define a region for the crosswalk to be placed 2) traffic lights, drawn in blue, defining a region to place them and where the starting colour will be randomly assigned (green or red) and will be changing periodically from green to red or vice-versa along the program run 3) Stop signs, drawn in red, to define a region where the car will stop for a brief moment before entering 4) Speed limit zone, drawn in cyan, to define a region of limited velocity 5) Pedestrian crossing where the user picks a point in the map for the pedestrian to start its walk and defines the orientation of the pedestrian movement, the instant of time at which he starts moving and the duration of the movement. Pressing 'I'm done!' exits this menu and if the user entered any speed limit zone he is asked to input the limiting velocity for those zones, so only one limit velocity can be used for all the speed limit zones.

From this, the user is now asked to define a path, needing to pick any number of points (and at least two) inside the previously defined roads, always pressing 'No' if the user is not finished in picking those points. In the end, the user is asked to input both orientations for the starting and ending points and ends this section by inputting a few more parameters like the total car energy in the beginning, the maximum velocity for the car and a threshold value to define the distance that one object as to be from the car to be counted as a collision. All of these parameters come with a default value, as a guide for the user to know what range of values should input.

If the user chooses in the beginning for a predefined image map, a list menu will appear for the user to choose the map and will iteratively pop-up menus asking the user if he wants to change the defined roads, road marks or path. Pressing 'No' in all of these menus will use all the predefined files made available by the developers.

The program will start to process all the information to define a path and account for the user inputs and after finishing will display the entire map in the left, a zoomed image of the car in the top right corner and the current velocity in the bottom right corner, displaying pop-ups along the program run when encountering some road mark or other events happen.

A flow chart of the initial input sequence, as described above, is available in Appendix 1.

IV. Guidance

As described in the introduction, the Guidance subset of a GNC system has the main goal of establishing the desired reference path to be followed by the autonomous car. The process of generating this path may be divided into three parts, where the first addresses the user needs and wants for a certain map environment and path by defining the used map, the points where the robot has to pass and some road marks to take into account in the establishment of the reference route. From this, the second part is to generate the reference, by dividing the map into a grid of points, so that the problem is simplified by not using each and every image pixel as a node to apply a shortest-path algorithm, namely a dynamic Dijkstra algorithm. Having a defined path, there is still the need to smooth it, by applying spline methods and interpolations so that there are no abrupt turns in the path, which would be unattainable by a realistic car.

A. Map Information

The first contact a user has with the developed software is through a simple graphical interface to enable the user to choose a map and input some important parameters so that the defined path, and therefore the car, acts as the user intends to.

1) Map Choice: In the beginning, the user has two options, as seen in Fig.1:

1) Defining an entirely new map, having to input the path to a new map image, saved inside a folder with the same name the image has, excluding the extension.
2) Choosing to use one of the predefined map images, selected through a *list dialog*

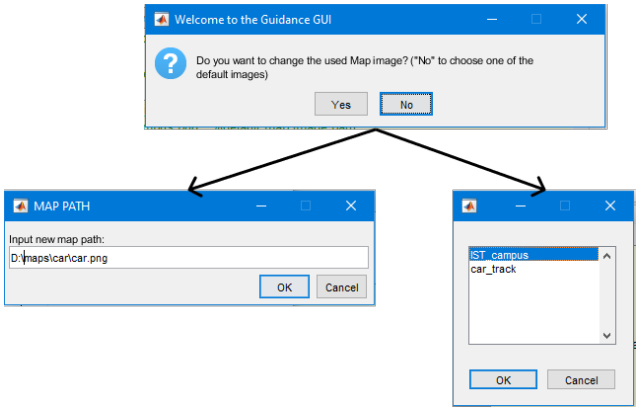


Fig. 1. Graphical User Interface for map image definition

After this, there is the need to define the relation between world meters and image pixels, the so called *meters-to-pixel ratio*. For this, the user has to choose first if his image is a simple sketch or a real context map, where for a sketch he will simply input some value for this ratio, where an input of 0.17 means that 1 pixel in the image is equivalent to 17cm in the real world, and for a real context image he will be asked to pick two points and the corresponding world geographical coordinates (latitude and longitude), which makes it easier thereafter to convert a world distance to a pixel distance, as seen in Fig.2. It is important to notice that this conversion is done with the consideration of a flat world, so the curvature of the Earth is not taken into account, mostly because the maps used are small enough to make that simplification. To note that for an already defined map, the user does not need to define again this meters-to-pixel ratio, because it is already stored.

B. Occupancy Matrix

Now that the user has defined the map to be used, an important step needs to be taken which is the development of an occupancy matrix. This occupancy matrix defines not only what places of the map the car is able to move but also where some expected road marks are placed, like crosswalks, traffic lights and stop zones, to have a more realistic simulation. For this, a simple car track map will be used as shown in Fig.3

1) Road Definition: The occupancy matrix starts by being defined as a matrix of zeros with the original dimensions of the image map used. From there the user is asked to draw polygons over the image where he wants the roads to be placed, defining those regions as *ones* in the occupancy matrix. When closing one polygon, the user is asked if he is finished defining the roads or not, where pressing "No" will enable him

Fig. 2. Graphical User Interface for defining coordinates in real map

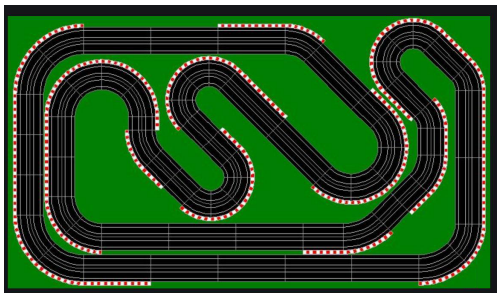


Fig. 3. Car Track map image

to continue on defining new roads. This process is shown in Fig.4. To note that drawing roads over already defined roads will have no impact in the occupancy matrix definition because in case of overlap that region will still have ones in the matrix.

Fig.5 shows a mesh plot of the occupancy matrix after defining roads, where one can see the difference between a road (elevated part) and a non-driveable place (lowered part).

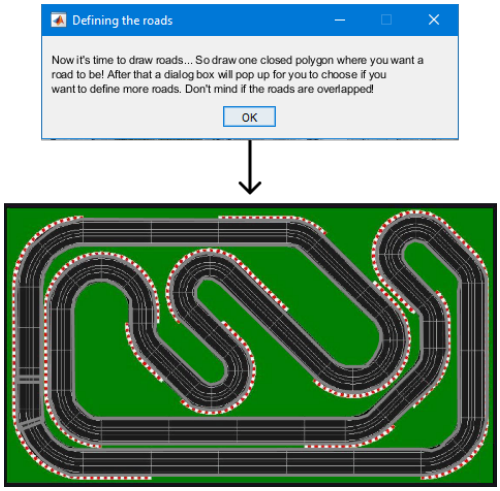


Fig. 4. Defining roads in the map

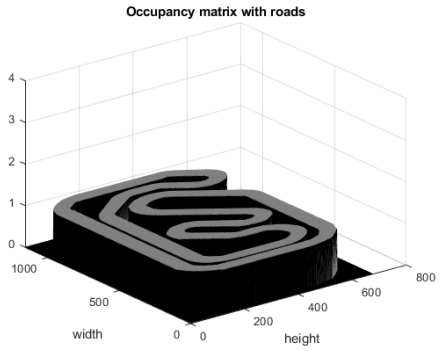


Fig. 5. 3D occupancy matrix only with roads

2) Road Marks: The next step is for the user to define some road marks that are already known and predictable even before the car starts its drive. Five road marks are available for the user to choose:

- Crosswalks: Where the car will slow down previously to entering one and where pedestrians can randomly traverse - Shown in white colour
- Traffic Lights: Where, based in sensor detection, the car will stop or continue depending on the traffic light colour that is currently on as it is constantly alternating between red and green - Shown here in green colour, but changed in the code to blue to not confuse the user as it could lead him to interpret it as always being a green passable sign.
- Stop Signs: Where the car will briefly stop and the reference path will be developed trying to avoid them - Shown in red colour
- Speed Limit Zones: Where the car will reduce its a velocity to a user defined limit - In cyan blue
- Pedestrian Crossing: Where the user defines the position of a pedestrian and the orientation, duration and initial instant of the pedestrian movement. The car will stop if detecting a person crossing the street or about to collide with the car.

The user defines these marks the same way roads were defined, by drawing polygons, as shown in Fig.6, and depending on the road mark, a number will be defined in the occupancy matrix. So a crosswalk region is defined as a ”2”, a traffic light is defined as a ”3”, a stop sign is defined as ”4”, the speed limit zones are defined as a ”7” and finally the pedestrians crossing does not need to be accounted in the occupancy matrix itself. The occupancy matrix with both roads and road marks is now defined as the mesh grid shown in Fig.7 with a few of the previously mentioned road marks, namely crosswalks, traffic lights and stop signs.

C. Graph

As previously mentioned, making computations such as shortest-path algorithms in a full image using the entire number of image pixels as nodes is unachievable, so a grid to diminish the number of nodes fed to the algorithms was needed. Two possibilities were explored where 1) was using a triangulation method and 2) making a quadrangular grid, with the last being the one chosen for the reasons explained below.

1) Triangulation vs Grid: The first approach, of triangulation, was to divide the drivable zone into triangles, using the triangle centroid has a node to use in the shortest-path algorithm. This method, although successfully implemented, had a few problems from the start:

a) First: Because a triangle only has three sides and they were done in such a way that the vertices were coincident with a road frontier, this meant each node only has two adjacent nodes, or so called neighbours which had a big impact on the freedom to create a path.

b) Second: Adding more nodes per triangle like using the centroid and and incenter point reduced the problem described above but created the problem of generating a very oscillating path because there was no pattern with the x and y coordinates of the nodes.

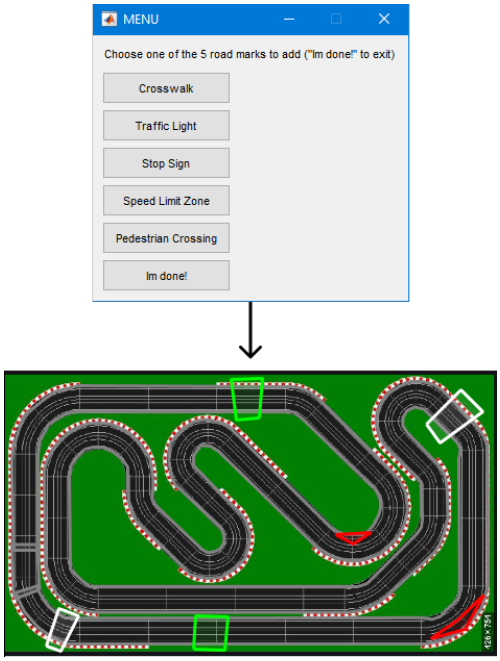


Fig. 6. Defining road marks in the map

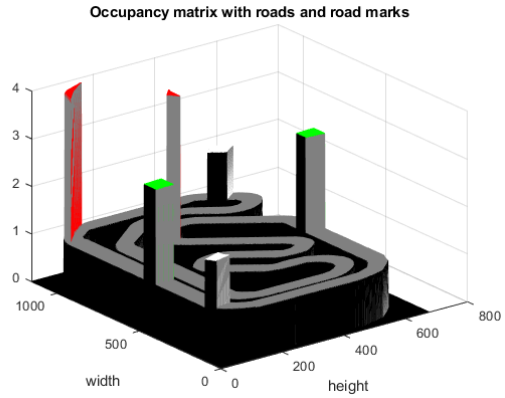


Fig. 7. 3D occupancy matrix with roads and road marks

To solve this, a quadrangular grid was implemented, where each node was equidistant to its eight neighbours. This eased the development of acceptable straight lines and added the capability of tuning the grid width parameter up if there was the need for more nodes and a more well defined path or tuning it down for a lower computational cost in developing the path. To note that this approach allowed, as it will be seen in IV-D2, a bigger control in the direction the path was intended to take, because each node always has eight fixed directions to move, equal for each and every node.

2) Point approximation to grid: Because points will almost never coincide with the drawn grid and the shortest-path algorithm only works with points from the grid (nodes), an approximation from the real point position to the grid points was made, as shown in Fig.8. This approximation is later corrected when smoothing the path, so that the reference path is as close as possible to the one given by the user.

D. Path planning

FOLLOWING some constrains, it is formulated a minimisation problem according to the time and mean velocity foreseen for the car visit

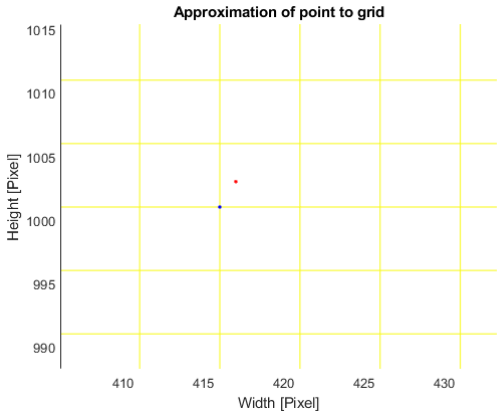


Fig. 8. Approximation of point to grid

the checkpoints. In this section the themes like the awareness of keeping the car away from the circuit limitations (e.g. sidewalks), the Dijkstra algorithm using dynamic weights (as in [1]) are covered.

1) Safetiness Matrix: One of the big risk that is faced when planning a path is when it does not avoid tight corners or segments, that would be an human choice by intuition. The proposed solution redefines the map and classifies the safetiness of driving in that area. Therefore, it is defined two thresholds:

a) Forbidden distance: Takes into account the dimensions of the car and an error of car position to classify areas as circuit limitation. This areas are those where it is impossible for the car to drive in, for example a tight road. The Fig. 9 represents a disk of radius 2 meters that corresponds to nearly 9 pixels in the image 3. Therefore, it is convoluted the circuit image with the disk, and all pixels that are closer than the threshold to a circuit limitation do not have the maximum convolution output though are not allowed to drive in. This transformation represented in 11 from the light blue to light green area.

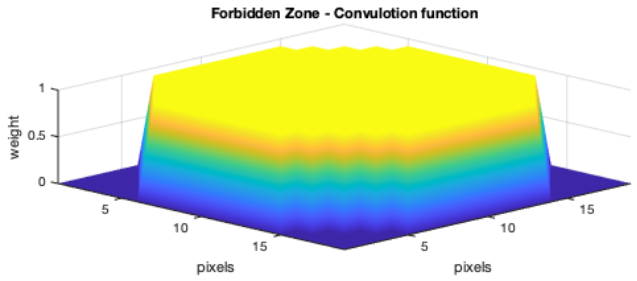


Fig. 9. Convolution function used to disallow all pixels that are closer than a threshold to a circuit limitation.

It is used a 2D convolution with a pair function centred in a pixel, in order to evaluate that pixel according to its neighbour as it is the mass centre of the car. This method also avoids collisions during curves, but disallows zones that would be feasible if the car were drive in parallel, since it uses the maximum size from the centre to the corner instead of the width.

b) Safe distance: Lastly, it is define a threshold distance that ensures the total safetiness of the car since it weights the closeness to a boarder. This areas, even though it is possible to drive in, it is less expansive (once it is safer) to drive in the middle of the road. It is used a logarithmic function in the range $[0, 1]$, and the 3D representation of the convolution function is in Fig.10.

To illustrate properly the transition threshold used is 3.5 meters although it might me too much in a real world since the road might not have twice that value in width, so in the next examples it is used a

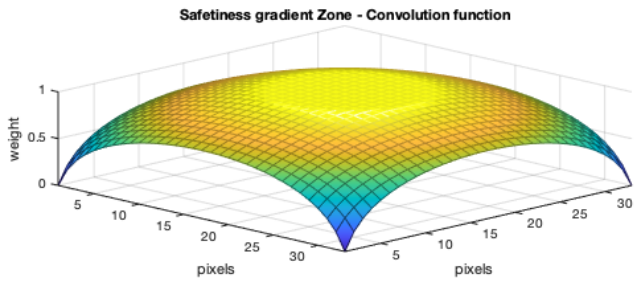


Fig. 10. Convolution function used to used to empathise the human behaviour to drive away from the borders.

lower threshold. The transition can be seen in comparison with the are on the middle and the area on the right of the Fig.11.

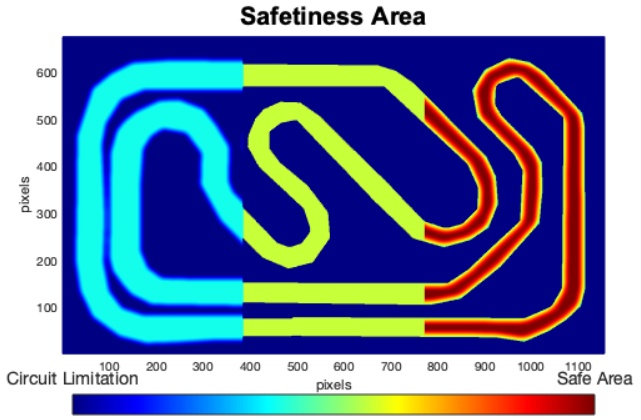


Fig. 11. Differences from the original road on the left, to the reachable area on the middle, and the preferable area for the car drive in on the right.

The route of the Fig.12 starts in the top point and its clear that the initial and final orientation are common and match with the pre-defined, and finally the algorithm is set to perform the curve by the interior like a human does. According to results of 12, it is evident the effect of the safe distance that repels the route from the borders and the forbidden distance shortens the width of the road [1].

Analysing the metrics in the Table I it is evident the increase of the distance with the increase of the threshold values. On the other hand, wider curves allows to maintain an higher mean velocity [2] and therefore it increases with the distance in this example. For both reasons above the duration and the speed [3] are practically identical.

TABLE I
SIMULATION METRICS FOR DIFFERENT PARAMETERS OF THE SAFETINESS MATRIX AND MAXIMUM VELOCITY OF 30 KM/H.

Curve	1	2	3	4
Forbidden distance	0.5	1.5	0.5	1.5
Safe distance	1	1	3	3
distance [m]	80.88	82.34	86.24	88.30
duration [s]	18.00	18.09	18.02	18.10
mean velocity [Km/h]	16.05	16.39	17.23	17.56
speed [Km/h]	5.96	5.94	5.96	5.93

[1]The width of the circuit straights is 16 meters.
[2]Mean velocity is the *route distance/time*.
[3]Speed is the *straight line distance/time*.

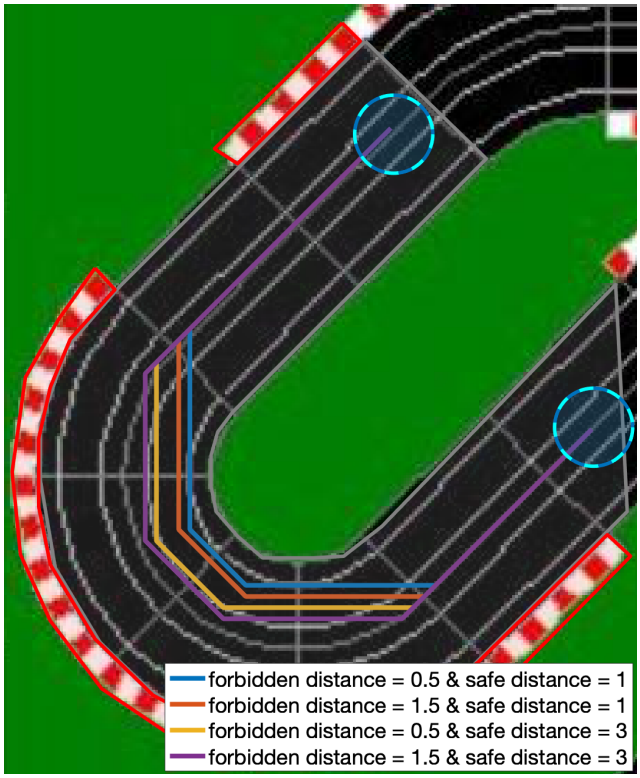


Fig. 12. Influence of the safetiness parameters in the path planning.

2) Dynamical Dijkstra: The algorithm implemented to computed the route between two points is based on the Dijkstra, although the weight for each points is not fixed since it depends on the historical of the route until the point (Dynamical Dijkstra). In exception of the nodes near the borders of the road in the safetiness matrix and the final point that as a fixed orientation, every point can be reached by any of its 8-neighbours under certain circumstances that are explained later. The prediction step, which is moves on the grid descried in the previous section, has fixed orientation according the 8-Point compass Rose, avoiding therefore jumps for other non-direct cell.

The algorithm computes the direct route between two points, joining all admissible together in the end. If an intermediate point cannot be reached because the collision risky is eminent or inevitable, it is skipped. On the other hand, if the first segment cannot be doable all path is classified as invalid, however if the last point impossible to be reach the route is planned until the previous. The reasoning is based on a real-situation once it might be better to redefined the original constrains before starting the simulation and on the other side, once the last point cannot be reached, the performance of the car can be tested as far as a admissible approach to the goal.

As an auxiliary data struct for the Dijkstra, it is used an heap (with minimisation criterion) since all computational cost are $\mathcal{O}(\log n)$ (e.g. sort, insertion) unaccepted the search $\mathcal{O}(n)$ and the extraction of the minimum value/zero level cell $\mathcal{O}(1)$. The cluster information associated to each cell stored in the heap is composed by;

- The lower **cost** to reach the cell;
- The *distance* from the staring sub-route point;
- An *index* to mapped from a 2D array to a 1D;
- The **previous** direction to reach the cell;
- The simulated **linear velocity** registered on the cell;
- The simulated **angular velocity** predicted on the cell.

The pseudo-code of the implemented algorithm is in the Table II. Firstly, once the heap is sorted the zero level element is always the cell with lower cost. The second step matches current cell with the last

TABLE II
PSEUDO-CODE OF THE DYNAMICAL DIJKSTRA ALGORITHM

	Initialisation: empty *if* Start Point *else* information about the last point of the previous sub-route.
1.	Get zero cell from the heap: $cell \longleftarrow Heap.head$;
2.	Exit validation: Return *if* cell is the final sub-route point.
3.	Identify new cell candidates: valid $if \angle(cell; candidate) \in \{-45°; 0°; 45°\}$
4.	Get new information:
4.1	*distance; index; previous direction*
4.2	*dynamical velocities*
4.3	*dynamical cost*
5.	Update/Insert in the heap the candidate: $Heap \longleftarrow candidate$

point of the sub-route and successfully stops the algorithm, the other exit options is when the number of elements in the heap is zero and means that the node is not reachable. The third steps computes the candidates cells with no more than a $\pm 45°$ deviation from the previous direction. This results is used to update the information of the candidates' cell in the first sub-step of the fourth step. The second sub-step is related to both linear and angular simulated velocities, and follow some constrains. The linear velocity is normalised to the unit value, and is not less than 0.05 units, when the car curves it maintains the previous velocity regarding that direction, therefore $v_n = v_{n-1} \cdot cos(\angle cell_{n-1}; cell_n = 45°)$. Finally, when the car goes on a straight line, the linear velocity increases 10% of the distance in meters of each step (this value was chosen according to the car parameters and the implemented controller). Finally the linear velocity is weighted bearing in mind the road marks, decreasing its value on the traffic light areas (green) and simulating very slow movement on the red areas (e.g. stop signs or congestion traffic). On the other hand the angular velocity is relive at each iterations, simulating a smooth recovery after a curve and summed with the new orientation difference. In the Fig.13 it is represent a route connecting both tower entrances int the south direction with intermediate points in both back corners of the central building. It is evident the slowdown on traffic lights area and the minimum velocity in the stop signs area (red). Moreover, it is also evident the continuous acceleration on straights lines (bearing in mind the 20 Km/h maximum velocity limit in the IST campus) and the slowdown in the curves.

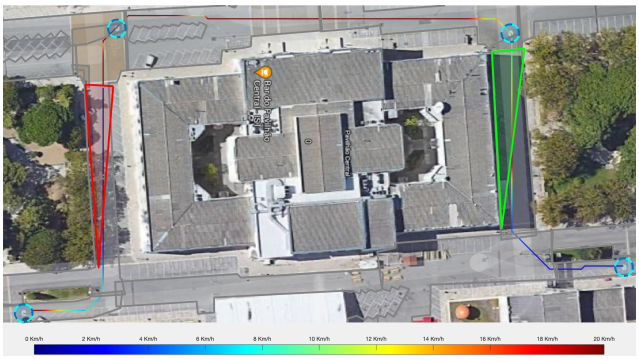


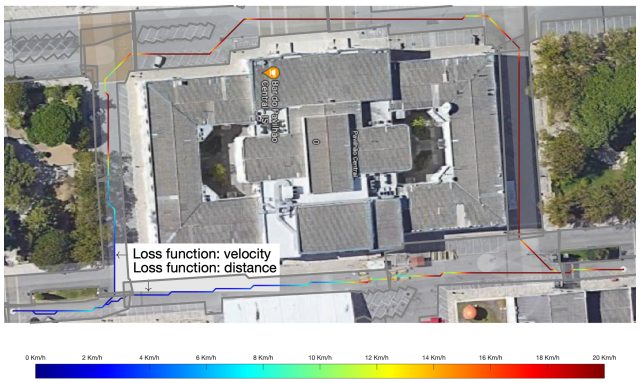
Fig. 13. Influence of road markers on the planned route.

The final sub-step is the computation of the cost value, where 2 different loss functions:

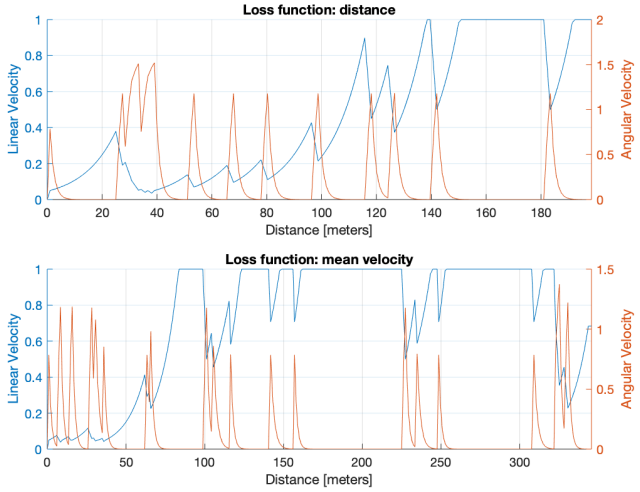
- distance: minimises the total distance of the route;
- velocity: minimises the angular velocity and maximises the linear velocity, according to:

$$Loss_{velocity} = \sum_{n=1}^{N} (1 - l.v._n)^2 + (1 + a.v._n)^2. \quad (1)$$

The cost value is the loss weighted in the safe matrix, for example, a cell in the middle of the road has the cost equal to the loss, although for a border cell the cost might be up to 200% of the loss value. The Fig.14.a shows the best routes starting in front the entrance of the south tower and ending in front of the entrance of the north tower in the IST campus. The best route without constrains follows a straight line between the two points although once the road in front of the central building is shorten due to the presence of a tent in that parking spot, the cost of the car to drive in there is much higher. Moreover all the adjustments curves to do in order to place the car in the middle of the road makes it harder to get a high linear velocity (see 14.b) and for both reasons the best route taking into account the quadratic velocity loss drives the car all around the central building. In the Table III are displayed the simulated metrics for each route, where it is highlighted the differences mainly on the distance and mean velocity according to each loss function. In this case the increase of 7 seconds allows the car to drive in a safer route, and the probability to avoid collisions are higher.



(a) Different routes from same the checkpoints.



(b) Simulated Velocities Linear(normalised to the unit) and Angular (radians).

Fig. 14. Best routes for both distance and velocity loss functions.

TABLE III
SIMULATION METRICS FOR DIFFERENT PARAMETERS OF THE SAFETINESS MATRIX AND MAXIMUM VELOCITY OF 30 KM/H.

Loss function	distance	velocity
Distance [m]	174.91	304.65
Duration [s]	68.20	75.96
Mean Velocity [Km/h]	9.23	14.44
Speed [Km/h]	9.00	8.08

To conclude, the initial points defining a route are converted to the closest cell in the grid, and the angles are mapped to the 8 available directions. For the initial distance, the algorithm only starts with the pre-defined orientation. However in order to meet the final orientation, it is disallowed any neighbours of the final point but the one that corresponds to the final orientation. At this points, the main disadvantage of using the algorithm emerges, and corresponds to the fact that the both the final point neither the previous, due to the final orientation force and the movements ($\pm 45^{\circ}$ constrains, cannot be reached, and therefore the end point has to be two cells before the final point in the inverse direction of the final orientation, since it is the last point that is outside the dome around the final point. This solution also gives more space to that the car reach the final point more smoothly with less orientation errors. The disadvantages of taking into consideration the initial orientation only on the rounded cells might introduce an error since the first step of the car is to reach the first point of the path (ideally the closest on the grid), which might not be in the same orientation of the desired initial orientation. The range of this error is in the order of the centimetres (maximum), since the worst case scenario is when the initial point is in the middle of 4 cells and therefore the maximum distance before starting the correct initial orientation is $(distance\ between\ cells)/\sqrt{2}$.

The final purpose of the guidance is to transpose if possible the route to the real world. As it was explained before, it is assumed a planar world which for example gives an error of approximately 1,13 cm error in 300 meters for the IST campus. It was used the *geoplot* MATLAB function [2], in order to test the computed coordinates and the result is displayed in the Fig.15. The results obtained are very accurate and the only miss shown are correct because the map incorporate in the function is not up to date (e.g. the unique road in front of the north tower).

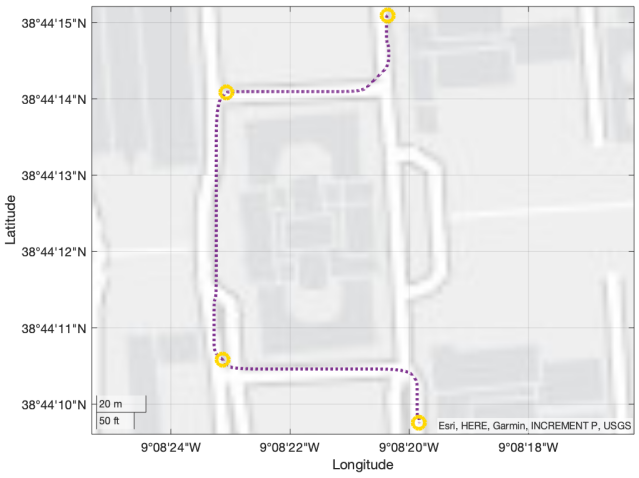


Fig. 15. Geographic coordinates from the route 13.

To conclude, the results obtained with this path planning implementation are positive, although it has space for improvements mainly on the 8 neighbours that could be double, using a 16-point rose compass reasoning. In addition more loss function could be added for example a duration or energy minimisation. This function does not require more parameters than the ones computed, although choice of using the velocity loss function boils down to the easiest route to be followed. Lastly the path planning implementation is program also to be recalled if the car is lost or if wants to change any checkpoints.

E. Path smoothing

THE path is now planned, however when it changes direction, that change is abrupt (with 45°), which is not feasible by a car. Due to that it is needed to make the curves smoother to simulate a more realistic path for the car. In order to make the path smoothing, it was made following 3 steps: putting the checkpoints in the path, divide it by segments and use the b-spline curves to make the path smoother.

1) Put Checkpoints in the Path: Due the fact that the checkpoints were approximated to points in the grid for the path planning, now it is necessary to put them back in the path in order to the car to pass in those points. So one changes the point closer to the checkpoint, in the array of points that compose the path, with the checkpoint coordinates.

2) Segmentation of the Path: Firstly, the algorithm searches the points where the path changes its direction, which this points are called *change points*. Then they are aggregated in groups according to a distance, that it is defined to be a reasonable distance to consider a point for the same part of the path, called *aware distance* which it was defined with a value of 10 meters. If the distance between the *change points* are bigger than that *aware distance* then the *change points* are not in the same segment of the path, otherwise they are in the same group, which it is called *cluster*. In Fig.16, it is represented the path and the *change points* that compose each *cluster*.

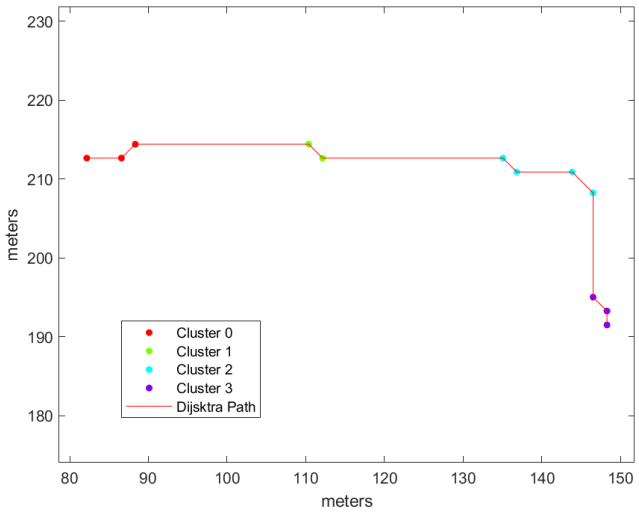


Fig. 16. Path segmented in clusters

3) B-Splines Curves: To do the smoothed path it is used the b-spline method (see [3]), with degree of 3, to approximate the path returned by the Dijkstra's algorithm to a fluid path. However, due to the fact that the b-spline tries to interpolate a function that approximates all the control points, in this case, all the points in the path, then some curves and information in the path might be lost. In order to fight that, it is used the b-spline approximation for each *cluster*, so the least information is lost.

In Fig.17, it is represented the path in the 16, after the b-spline interpolation for each cluster. The curve of the *cluster* 2 is zoomed in for one to see the approximation and curves made which makes the path smoother. An important detail to talk about is that if there is a checkpoint, the b-spline curve tend to approximate more to that point, because it has more weight in the interpolation than the other points, in order to be sure that the car passes in that checkpoint.

Finally the guidance group also designed the car based on the real measurements (see Table IX). Firstly it was designed a 2D model of the car and than rotated and translated based on the orientation of the car and the steering wheel. This model is useful not only to the real-time demonstration of results but also because it helps to understand better (by visualising) the trajectory followed by the car.

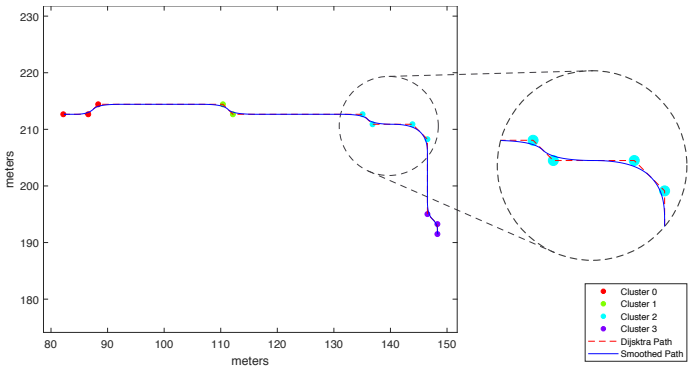


Fig. 17. Smoothed Path.

V. Navigation

The Navigation Segment of this work is divided between the Tracking Problem and the Environment Dynamics, where both are deeply connected in order to make the Autonomous Car capable to handle any environment and to reinforce its position estimate using sensors with low error.

In the Tracking Problem, the main focus of the group was to provide a refinement of the position estimate that came from the Control group, which comes with an added error since it is constructed through the wheel spin and velocities. In order to fulfill our goal, we refine the estimate using an *Extended Kalman Filter* in combination with a Measurement Model that contains sensors such as *GPS* and a *Orientation Sensor*.

For the Environment Dynamics, the main challenge is to simulate other sensors like a *camera*, *lidar* and *proximity sensors* to detect the surroundings. Capturing the environment, as we humans do with our senses, is essential for cars to be able to drive autonomously. These three sensors detect and identify different objects that can appear during the car's trajectory.

A. Sensors

The *Navigation* component of the project relies on sensor information that we used not only on the Prediction Model, that refines our estimate position that comes from the *Control* unit but also to allow the autonomous car the ability to cope with a changing environment that has spontaneous variables such as, sudden objects on the track, People Crossing Roads, normal traffic light, which can directly affect the vehicle trajectory.

In order to cope with all these variables, we have chosen specific sensors that will be placed on the vehicle. This will be further explained in another section.

- **GPS** - The GPS provides accurate measurements of the position vector of the car. This will be used in the measurement model of the *EKF*. The GPS, depending on the signal interference might have sudden Breakups which will be considered throughout the experimental part.
- **Orientation Sensor** - The Orientation Sensor will provide the orientation that will later on be used in the measurement model of the *EKF*.
- **Visual Odometer** & **Odometer** - Provides Odometry measurements for Position and Orientation that are based on the analysis of sequential camera images (Optical Flow). The Odometer corresponds to the simulation made by the *Control* Group and gives the position estimate that comes from the wheel spin.
- **Camera** - Use visible light to capture the environment surrounding them. In a real-world situation, cameras automatically detect objects, classify them, and determine the distances between them and the vehicle. We will use the camera only to detect and identify different types of objects.
- **LIDAR** - Stands for "light detection and ranging". The sensor acts just like a sonar that utilises pulsed laser waves to map the distance to objects detected in the range of the sensor.
- **Proximity Sensors** - A proximity sensor is a non-contact sensor that detects the presence of an object when the target enters the sensor's field. This sensor is going to be used to count the number of collisions through the simulation.

There is a significant debate whether cameras are better or not than LIDARs. One of the strongest arguments that favored the use of cameras is the way we humans perceive our environment when we are driving a car. Therefore it only makes sense that robots use the same capabilities as humans have to achieve similar or better results, thus the use of cameras makes a lot more sense. Even more, cameras are significantly smaller and cheaper than LiDARs. They also see in better resolution and color, meaning they can read traffic lights and signs. However, LIDARs use infra-red light which, in theory, means that they will be less responsive on days where it is raining, fog, or other weather conditions, that are very unpredictable.

B. Tracking Model

The navigation component was implemented based on an *Extended Kalman Filter*, which is an extension of the normal *Kalman Filter* for nonlinear models and is used for prediction estimation for nonlinear transformations. In order to explain this, the first step is to illustrate the standard *Kalman Filter*, understand it's strongest and weakest points in this model and why is it required to have a transition to the extended version.

C. Kalman Filter

The group *Ribeiro et al.* [4] defines the *Kalman filter* as a linear, discrete time, finite dimensional time-varying system that evaluates the state estimate that minimises the mean-square error. In a tracking scenario, such as the one used in this project, the objective is to obtain a state prediction based on the past estimate and later refined by a given measurement model.

The *Kalman Filter*, as it can be seen from Fig.18, is constituted by the Prediction Phase, were the model, using the prior knowledge of the state, is propagated to the next state by using a given dynamics model. Then there is the Update Phase where the estimate from the prediction stage is refined using the measurements at that instant.

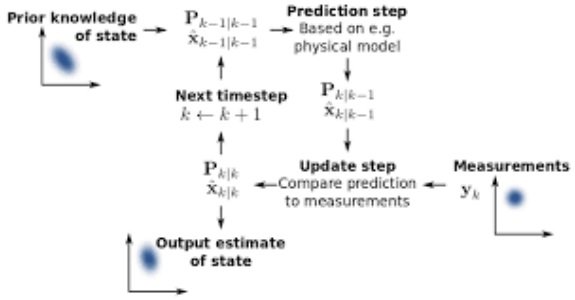


Fig. 18. Workflow of the Kalman filter

This method constituted our first trial at the *Navigation* problem, however we soon realised that this method required a linear time varying dynamics model to perform equation IV of the prediction step, which does not correspond to the Car model that will be seen in the next section.

TABLE IV
KALMAN FILTER - PRED. PROPAGATION EQUATION

$x_{[k]}^{-} = A_{[k-1]}x_{[k-1]} + B_{[k-1]}u_{[k-1]}$	Prediction Propagation Equation
$x_{[k]}^{-}$	State Estimate from Pred. Stage
$x_{[k-1]}$	Past State Estimate
$A_{[k-1]}, B_{[k-1]}$	Transition matrices
$u_{[k-1]}$	Control Vector

D. Extended Kalman Filter

In order to coup with the nonlinearities of the Dynamics Model of the Car model, it was implemented the *Extended Kalman Filter*. This method manages to be able to work with non-linear systems since it approximates the state to the 1^{st} order Taylor expansion and then uses this estimate for the update state equation, as it is seen from the equations V. [6]

TABLE V
EXTENDED KALMAN FILTER - GENERAL EQUATIONS

$\hat{x_k}^{-} \approx f(\hat{x_{[k-1]}}, \overline{v}), \hat{y_k}^{-} \approx h(\hat{x_k}^{-}, \overline{n})$	Taylor Expansion Approximation.
$\hat{x_k} \approx \hat{x_k}^{-} + K_k \cdot [y_k - \hat{y_k}]$	Update Equation.
x_k^{-}	State Estimate from Pred. Stage.
x_k^{-}	State Estimate from Update Stage.
$x_{[k-1]}$	Past State Estimate.
$\hat{y_k}$	Measurement Estimation.
y_k	Measurements.
$f(\hat{x_k}, \overline{v})$,$h(\hat{x_k}, \overline{n})$	System Dynamics and Measurement Matrices.

Before going into detail regarding the *EKF*, the Dynamic and Measurement will be displayed to illustrate the conditions of the model.

1) Dynamics Model: The state equation be defined by the polar coordinates, more specifically $[x_k, y_k, \theta_k]$ and the equation describing the movement can be described by the equations 2.

$$f(x_k, y_k, \theta_k) = \begin{bmatrix} x_{k-1} + N\cos(\theta_{k-1}) \\ y_{k-1} + N\sin(\theta_{k-1}) \\ \theta_k^{Odom} \end{bmatrix} \tag{2a}$$

$$F_k(x_k, y_k, \theta_k) = \begin{bmatrix} 1 & 0 & -N \cdot \sin(\theta_{k-1}) \\ 0 & 1 & N \cdot \cos(\theta_{k-1}) \\ 0 & 0 & 1 \end{bmatrix} \tag{2b}$$

$$N = ||x_k^{Odom} - x_{k-1}^{Odom}|| \tag{2c}$$

It is important to note that we consider having always available the *Visual Odometry* of the car and, this is calculated based on a technique called *Optical Flow*. This technique provides the variation of the car Position and Orientation based on the distribution of apparent velocities of movement of brightness pattern in an image, for example, the sensors could be associated with the cameras that are placed throughout the car. In this case, it can be simulated by $\theta_k^{Odom} = \theta_k^{GPS} + w_k$ where w_k corresponds to the error associated to the *Odometry* information. The variable N is determined with the variation in odometry that comes from the *Control* Group.

2) Measurement Model: For the measurement equations to get a measurement prediction, two different sets of equations were tested and can be seen in equations 3 & 4, where in the first case, our logic was to represent the position with polar coordinates in a 2-D plane since in our opinion we would get the most realistic results. For this reason, we have the norm of the position vectors and the orientation. In the second model, the normal absolute sensors, both GPS and Orientation Sensor, were used and defined a subsequent Gaussian error.

$$h_1(\hat{x_k}^{-}, \hat{y_k}^{-}, \hat{\theta_k}^{-}) = \begin{bmatrix} \hat{\rho_k} \\ \hat{\theta_k} \end{bmatrix} = \begin{bmatrix} \hat{\rho_k}(\hat{x}_k, \hat{y}_k) \\ \arctan_2(\frac{\hat{y}_k - \hat{y}_{k-1}}{\hat{x}_k - \hat{x}_{k-1}}) \end{bmatrix} \tag{3}$$

$$h_2(\hat{x_k}^{-}, \hat{y_k}^{-}, \hat{\theta_k}^{-}) = \begin{bmatrix} \hat{x}_k \\ \hat{y}_k \\ \hat{\theta}_k \end{bmatrix} \rightarrow \begin{bmatrix} x_{GPS} + N(0, \sigma) \\ y_{GPS} + N(0, \sigma) \\ \theta_{OrientationSensor} + N(0, \sigma) \end{bmatrix} \tag{4}$$

Therefore the Jacobian Matrix $\left(\frac{\partial h(\hat{x_k}^{-}, \hat{y_k}^{-}, \hat{\theta_k}^{-})}{\partial(\hat{x}, \hat{y}, \hat{\theta})}\right)$ is given by:

$$H_1(\hat{x_k}^{-}, \hat{y_k}^{-}, \hat{\theta_k}^{-}) = \begin{bmatrix} \frac{\hat{x}_k}{\sqrt{\hat{x}_k + \hat{y}_k}} & \frac{\hat{y}_k}{\sqrt{\hat{x}_k + \hat{y}_k}} & 0 \\ \frac{-(\hat{y}_k - \hat{y}_{k-1})}{\hat{x}_k^2 + \hat{y}_k^2} & \frac{\hat{x}_k - \hat{x}_{k-1}}{\hat{x}_k^2 + \hat{y}_k^2} & 0 \end{bmatrix} \tag{5}$$

$$H_2(\hat{x_k}^{-}, \hat{y_k}^{-}, \hat{\theta_k}^{-}) = eye(3) \tag{6}$$

Both models were implemented to see the best version and through extensive testing, model 2, with absolute sensing and added Gaussian errors, achieved better results especially in the scenario where the path was long and with more cumulative errors through time.

Once we have defined the complete specifications of the car Model, we can now start to understand the *Extended Kalman Filter*. Firstly, it is important to take note that as it was said before, the *EKF* uses the 1^{st} order-Taylor expansion around the estimated state to transform the non-linear system into linear equations, which was illustrated mathematically by table V and can be better seen in Fig.19 where the extend of this linearization can be better seen.

In addition, Fig.19 also showcases that the results from this linearization are reasonably close to the true value gaussian when the non-linear behaviour of the functions is close to linear and continuous form, since the Taylor expansion used belongs to a linear process.

In comparison to the *Kalman Filter*, the main process is fairly similar apart from the state propagation equation and the measurements equations, since now that system is no longer linear, as it can be seen by table VII.

TABLE VI
EKF NOTATION

Nomenclature	Connotation
f	System Dynamics
N	Distance Covered Between Iterations
P	Estimate Uncertainty
R	Measurement Uncertainty
Q	Noise Variance
H	Measurement Jacobian
F	State Jacobian
K	Kalman Gain

TABLE VII
EXTENDED KALMAN FILTER

	Initialisation: *a priori* values
1.	Prediction:
	$\hat{x}_{k\|k-1} = f(\hat{x}_{k\|k-1})$
	$P_{k\|k-1} = FP_{k-1}F^T + FQF^T$
2.	Update:
	$K_k = P_{k\|k-1}H^T \cdot [HP_{k\|k-1}H^T + HRH^T]^{-1}$
	$P_k = (I - K_kH) \cdot P_{k\|k-1}$
	$\hat{x}_k = \hat{x}_{k\|k-1} + K_k \cdot [y_k - h(\hat{x}_{k\|k-1})]$

Both in the *EKF* and the *Kalman Filter* there is a separation between the estimates from the normal prediction step and their respective corrected versions where, in this project, the connotation used was $\hat{x}_{k|k-1}$ for the state estimate from the Pred. stage and $\hat{x}_k$ for the state

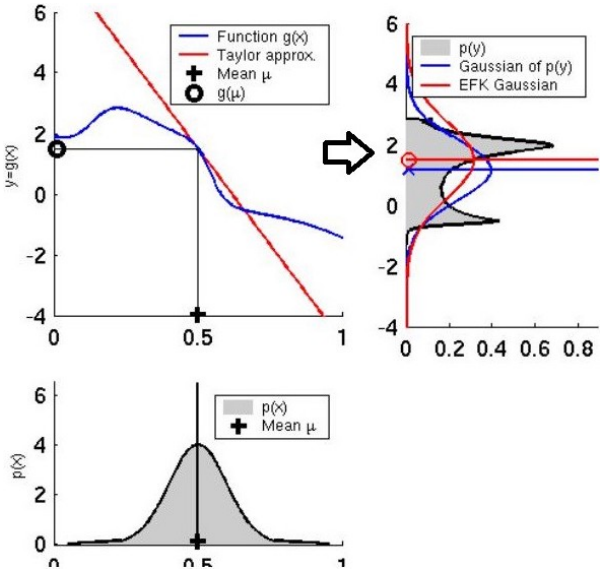


Fig. 19. Illustrative version of the *EKF* 1^{st} order Taylor Series Approximation

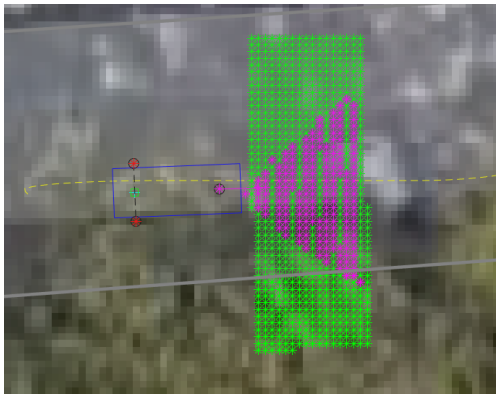


Fig. 20. Car sensors: the camera is represented in green and the lidar in pink

estimate that has been refined by the filter. The same notation applies to other variables in the filter.

In conclusion, the *Extended Kalman Filter* evaluates how feasible the data is compared to its knowledge about the system through sensors. This method computes the state estimate that minimizes the mean-square covariance error. Moreover, uncertainties (P and R) define the Kalman gain that in collaboration with the prediction state and the measurements, contributes to the updated state. For example, taking into account the case when the estimated uncertainty goes to 0, the Kalman gain is 0 and, therefore, the next state relies on the prediction of the model and, vice versa for the case when the measurement's uncertainty is small the Kalman gain matrix tends to the inverse of the observation's jacobian and so the next state relies on the measurements.

It is also relevant to note, that to make the simulation a better replication of the real-life scenario, Gaussian noise $N(0, \sigma)$ was also added to the final estimation since it would replicate better the output errors from a real environment with mostly gaussian error.

E. Environment Dynamics

ALL autonomous cars need sensors that will help them make sense of objects in the road, traffic lights, crosswalks, pedestrians crossing a street, just like human drivers do with their eyes. For this work, we will implement a LIDAR and a camera to help with the problem in question. To implement a simulation of a LIDAR and a camera in an environment like *Matlab* several steps had to be made. Fig.20 shows the vehicle with the sensors implemented, a camera in green and lidar in pink.

a) ***Camera****:* The problem of implementing a camera in an environment like *Matlab* is not as straightforward as capturing images of the surroundings and making sense of them to identify and classify certain objects. Therefore to make an appropriate sensor we had to change our approach.

That said, in essence, the idea is to make the camera a set of points, which origin is at the front of the car. The camera analyses the occupancy matrix and identifies objects according to different values in the matrix cells. Having said that, we will make a more detailed analysis of how we implemented the camera:

- The camera has a maximum range of 3 meters in the horizontal and 8 meters in the vertical.
- Each point in this set has a distance in the horizontal and vertical axis from the adjacent point of 0.1763 m(resolution of the map of the IST Campus). In that way, we can have a point in each pixel of the map within the range defined.
- To each point within the set, we apply a rotation followed by a translation. By applying such transformations, we can place the sensor in the front of the car.

For the purpose of clarity, we will discuss how the rotation and the translation are applied to these sets of points. First, we defined the rotation matrix for the rigid transformation of the camera and lidar according to the orientation of the car. The matrix that represents this rotation matrix, R is the following, where θ_{car} represents the orientation of the car,

$$R = \begin{bmatrix} cos(\theta_{car}) & -sin(\theta_{car}) \\ sin(\theta_{car}) & cos(\theta_{car}) \end{bmatrix} \tag{7}$$

On the other hand, the translation is obtained taking into account the geometry of the car and, since we want to place our sensor at the front of the car, this translation must be done according to the car's movement. The x and y values in Fig. 29 are already determined. However, these points relate to the center of mass of the car, and not the front.

The objective is to discover the values of x' and y' represented in Fig.29. By the Pythagorean theorem, we can immediately conclude that:

$$x' = x + dim \cdot cos(\theta) \tag{8}$$

$$y' = y + dim \cdot sen(\theta) \tag{9}$$

Finally, after defining the translation and the rotation needed, we can write the all transformation in a single equation as following:

$$[x_{Sensor}, y_{Sensor}] = [set\ of\ point] \cdot R + [x', y'] \tag{10}$$

By defining the set of points, as previously explained, we can then, at a given position of the car, go through all those points and inspect the value of the occupancy grid, and "see" if there is any traffic light, crosswalk, etc that the car might encounter.

b) ***Traffic Light****:* The traffic lights are represented, by *Guidance*, in the occupancy matrix by the number 3. Additionally, we made the traffic light that changes from red to green. The color of the traffic light changes every 10 seconds from red to green.

In this case, the camera will detect if, at any point covered by the range of the camera, the occupancy grid is equal to 3. Then, if a traffic light is detected, we will check the color of the traffic light. If it is green, the car will continue its path and, no display will appear, otherwise, a flag, *flag_red_light*, will be sent to the *Control* group to inform that the car has to stop.

Also, when the red light is detected, the display presented in Fig.21 is made to inform the user about the surroundings.

*c) **Crosswalk**:* The crosswalks are represented, by *Guidance*, in the occupancy matrix by the number 2. In this case, the camera will detect if, at any point covered by the range of the camera, the occupancy grid is equal to 2. If detected them a flag, *flag_passadeira*, will be sent to the *Control* group.

Also, when a crosswalk is detected, the display presented in Fig.21 appears to inform the user about the surroundings.

*d) **Stop Signal**:* The stop signal is represented, by *Guidance*, in the occupancy matrix by the number 4. In this case, the camera will detect if, at any point covered by the range of the camera, the occupancy grid is equal to 4. If detected, then a flag, *flag_stopSignal*, will be sent to the *Control* group to inform that the car has to stop for 2 seconds.

Also, when the stop signal is detected, the display presented in Fig.21 is made to inform the user about the surroundings.

*e) **People**:* Each person is represented in the occupancy matrix by the number 5, this attribute was made by the *Navigation* group, and will later be explained in section *People Path*. In this case, the camera will detect if, at any point covered by the range of the camera, the occupancy grid is equal to 5. If detected then a flag, *flag_People*, will be sent to the *Control* group to inform that the car has to stop because a pedestrian has been detected. When the camera stops seeing the pedestrian, the flag, *flag_People*, changes to zero, and we send this information to the *Control* group so that they can continue their path.

Also, when a person is detected, the display presented in Fig. 21 is made to inform the user about the surroundings.

*f) **Speed Limit**:* The speed limit zone is represented, by *Guidance*, in the occupancy matrix by the number 7. In this case, the camera will detect if, at any point covered by the range of the camera, the occupancy grid is equal to 7. If detected, then a flag, *speedlimit_signal*, will be sent to the *Control* group to inform that the car cannot exceed the speed velocity that the user specified.

Also, when the speed limit zone is detected, the display presented in Fig.21 is made to inform the user about the surroundings.

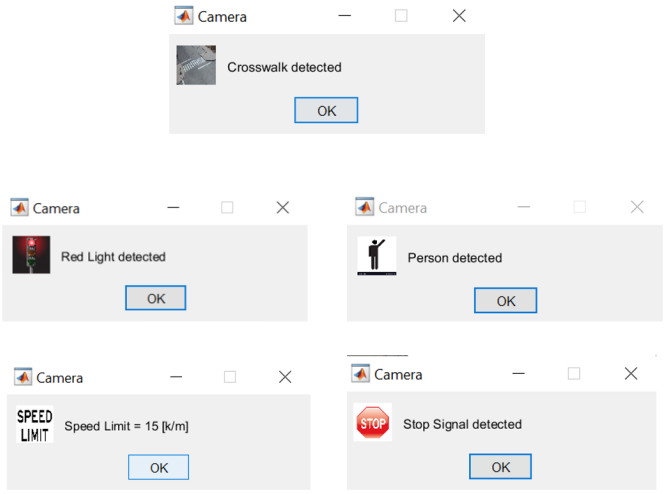


Fig. 21. Display of all detection's made by the camera

*g) **Lidar**:* The Lidar sensor uses the same approach as the camera. This sensor is made of a set of points in space, as represented in Fig.20, and has a maximum range of 2.75 and 4.8 meters in x and y dimensions, respectively. This sensor is at the front of the car in the middle and, the set of points perform a rigid body transformation with the car's movement. The Lidar detects an object if the occupancy matrix has a cell with the value 0 or 5 and a Lidar beam, besides it also determines the distances between the objects and the car. The distance is simply given by equation 11 . In this equation, the variables x_car and y_car are the values of the point at the middle of the car's front at a certain iteration.

$$distance\_to\_object = \sqrt{(x_{car} - x_{object})^2 + (y_{car} - y_{object})^2} \quad (11)$$

*h) **Inherent Collision**:* In some situations, the car may collide with an object or deviate from the path and get out of the road. To prevent these situations, the Lidar predicts the possibility of a collision. In other words, whenever an object is detected, the linear velocity model is used to check a collision between the car and that object in the following ten iterations. For each iteration, the car's and object's position is updated with equation 12. $Velocity_x$ and $Velocity_y$ are the values of the velocity that the car or the object is at the moment of the detection and, 0.1 is the time between two iterations.

$$x_{new} = x_{initial} + 0.1\ .\ velocity_x\ .\ n_{iteration} \quad (12a)$$

$$y_{new} = y_{initial} + 0.1\ .\ velocity_y\ .\ n_{iteration} \quad (12b)$$

$$threshold = 1.0630 - (1.0630/(11 - n_{iteration})) \quad (12c)$$

After determining the new position for the car and object, the distance between these two is calculated. If this distance is below a certain threshold, the car will, eventually, collide with the object if it continues with the same velocity. For this case, the threshold depends on the number of iteration. Initially,it is 1.0630 meters. This value decreases with each iteration since the car is approaching the object. At any iteration, if the distance is smaller than the threshold, it activates a flag $flag\_Inerent\_collision$ to warn the *Control* group of a future collision.

*i) **Proximity sensors - Collisions**:* The number of collisions is determined with proximity sensors since these sensors can detect nearby objects without any physical contact. At the beginning of the program, the user chooses the value of a threshold that limits an area around the car. If an object is at any moment in this area, the program detects a collision. In other words, if the occupancy matrix contains a 0 or 5 around the car, then the number of collisions is incremented by one. This number is displayed during the simulation as depicted in Fig.22. The first Figure represents the display during the simulation and the second at the end of the simulation.

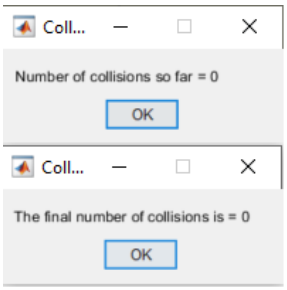


Fig. 22. Display of number of collisions during the program

F. GPS Breakups

In this project it was made a priority to replicate as much as possible the real-life situations. For this reason, we have considered the *GPS Breakups* which are zones where there is an interference in the GPS Signal, meaning that it is impossible to access these measurements in a short period of time. In the general case, since the concept was to implement for different maps, these GPS Breakup Zones are random where the probability of appearing can be defined by the user, and appear in the time plot at the end of the program. Moreover, in the main map, which is the *IST Campus*, the *GPS Breakups* zones are also pre-defined

to the zones closer to the North and South Tower, as seen in Fig.23 , which are clear instances where in a real-life scenario it would cause this situation.

In these scenarios, the normal *Extended Kalman Filter* only propagates the model, in other words, it only uses the prediction stage without any type of sensor refinement.

Fig. 23. Example Area of GPS BreakUps near IST Towers (Orange)

G. Energy Based Maximum Velocity

In this section we will discuss the implementation of the conservative estimate for the maximum velocity that allows the car to reach the end of the mission. For that purpose, we will suggest the following algorithm.

For each iteration, we will ask for the available energy for the remaining of the trajectory, *E_budget*, to the *Control* group. We also set a constant P_0 to *1000 W*, this is the power that is consumed by the car when it is stopped. The value of Δt is 0.1 s, the time set between consecutive iterations.

As we receive the information about the variable, *E_budget*, we do an immediate verification to determine if there is any energy remaining for the car to move. If there is not we send a flag, *flag_energy*, to the *Control* group to inform them that the car has to stop.

Then in order to start computing the maximum velocity we start by determine the available energy per step,

$$\frac{E\_budget}{N_{steps\ remaining}} \tag{13}$$

To calculate the variable $N_{steps\ remaining}$ we need to make an estimate of the total number of iterations needed for the car to follow the predefined path. That said, first we will need to calculate the total distance of the path based on the information receive by the *Guidance* group,

$$total\ distance = size\ of\ x_t \cdot fixed\ sample\ rate \tag{14}$$

Having that distance we can simply determine the number of total steps, assuming that the car always move with the minimum velocity, $1m/s$,

$$total\ number\ of\ steps = \frac{total\ distance}{v \cdot \Delta t} \tag{15}$$

In view of this information the variable $N_{steps\ remaining}$ will just be the difference between the total number of iterations and the current number of iterations at any time during the simulation.

Finally we can calculate the maximum velocity for each iteration taking into account the following equation,

$$maximum\ velocity = avail\ energy\ per\ step/\Delta t/P_0; \tag{16}$$

H. People Path

In this section, we will discuss how the pedestrian's paths are made. There are two situations to take into account in the laboratory statement. The first one is that the pedestrians should walk on crosswalks, these are initially defined at a fixed location. In the second situation, the path of the pedestrians is set by the user when the program starts.

The paths are made in different ways for the two situations:

- In the first situation, the pedestrians walk in horizontal or vertical paths and they always appear on crosswalks when the car is approaching
- In the second situation, the *Guidance* group asks about how many pedestrians does the user wants in the simulation, their location, at what point should they appear, and for how long should they walk. Additionally, it is also asked for the orientation of the pedestrians.

The time defined by the user is set to a number of iterations. When this number of iterations is reached in the simulation the pedestrian appears. The number of iterations is calculated by the following equation:

$$iterations\ people = \frac{duration\ of\ the\ crossing}{0.1} \tag{17}$$

After requesting all parameters needed, we can make the algorithm that sets the pedestrian's path in the second case. By default, the pedestrians are set to walk 20 meters. Within a while loop, the final position of each pedestrian is updated for every iteration. For that purpose, we first calculated the velocity of the pedestrian taking into account the distance and time that they walk with the following equation:

$$velocity\_pedestrian = \frac{distance\ of\ 20\ meters}{duration\ of\ the\ crossing} \tag{18}$$

Given that we also ask for an orientation, we additionally have to calculate the velocity in the x and y axis in the following equations:

$$velocity\_pedestrian\_x = \frac{distance\ of\ 20\ meters}{duration\ of\ the\ crossing} \cdot cos(\theta_{people}) \tag{19}$$

$$velocity\_pedestrian\_y = \frac{distance\ of\ 20\ meters}{duration\ of\ the\ crossing} \cdot sin(\theta_{people}) \tag{20}$$

Having the velocities defined, we can simply take use of the following equations to calculate the final position of the pedestrian for each iteration:

$$final\ position\ x = previous\ final\ position\ x + velocity\ pedestrian\ x \cdot 0.1 \tag{21}$$

$$final\ position\ y = previous\ final\ position\ y + velocity\ pedestrian\ y \cdot 0.1 \tag{22}$$

When the difference between the initial and final position is bigger than the default distance then the loop breaks and the pedestrian's path ends.

In the first case, the pedestrian path is defined by another function inspired by the function created by the *Control* group, called *robot simulation*. The only difference in this function is that the orientation of the pedestrians is set to 0, and they walk 15 meters in 5 seconds. One other aspect to take into account is the four possible outcomes for the pedestrians crossing.

Pedestrians can walk vertically or horizontally as previously said. This differentiation is based on the orientation of the car. If the car is moving vertically, which is similar to saying that the $sin(\theta_{car}$ is less or equal to 0.5 then the pedestrian should walk on the crosswalk in the horizontal. If instead, the car is moving horizontally, by the same logic, the pedestrians should walk vertically.

Also, we always guarantee that, in the first situation, the pedestrians always appear in front of the car. To make that verification, we have to ensure that when the car is moving vertically, it comes from the bottom up, or from the top down on the map. Likewise, when the car is moving horizontally, we have to make sure it comes from the left to the right or the right to the left on the map.

I. Navigation Simulation

In this segment, we intend to perform two important experiments to our individual model in order to test its ability to coup with measurement error, which is a strong characteristic of the real-life scenario, and the convergence capability. In order to do this, we tested our model firstly with a path that represents the letter *'Z'*, since this letter contains sharp edges and long straights, which represents a difficult scenario for any car tracking model. The tests performed, involved normal errors defined in Table VIII, Strong errors applied to the measurements to showcase the ability of the filter to coup with low-quality sensors, which is a better representation of the real-life scenario and the scenario where the model starts with an incorrect position that may come from GPS Breakups at an early stage and can test the ability of the model to converge to the true path.

It is also important to take note that we have implemented the *MATLAB* version of the *Extended Kalman Filter* specifically for tracking scenarios [10]. This version was placed on every experiment in order to serve as the theoretical model of our method.

TABLE VIII
EXPERIMENT TABLE OF INFORMATION

Connotation	Value
Normal Measurement Error	0.001% of y_n
High Measurement Error	0.01% of y_n

1) Test with Normal Error: As it was previously stated, the first experiment intends to replicate the scenario with normal simulation conditions where the error was obtained from other papers such as *wang et al.* research group [9]. In general, from Fig.24 it is possible to see that the method obtained had reduced errors and close to the order of the applied sensor errors (10^{-4}). Moreover, it can also be seen that our model maintained in the same order as the *EKF* from *MATLAB*, reaching an overall performance that is quite similar to the theoretical one.

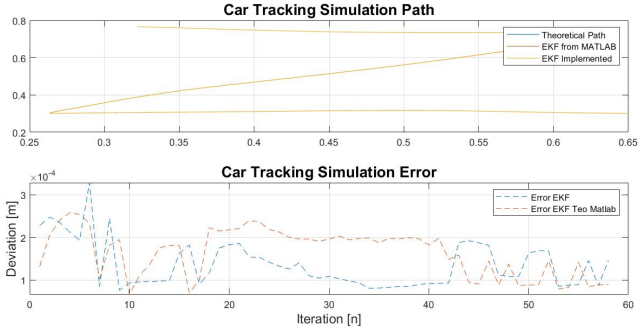


Fig. 24. *'Z'* letter with Normal Conditions

2) Test With High Measurement Error: In the second experiment, the objective was to understand clearly how the *Extended Kalman Filter* would work in a more realistic scenario, when the errors on the sensors are quite high, representing the situation of cheap sensors or even a more noisy environment for them. From Fig.25 it is possible to understand that the overall error increased and both filters maintained a high level of convergence to the theoretical path showcasing their ability to handle a real-life scenario where there is a predominance of noisy sensors.

In addition, in this case, even though in some instances the proposed implementation of the *EKF* had a better performance than the *MATLAB* version, the overall score, in this, the situation was better in the Theoretical version, meaning that this would be better in a real-life scenario.

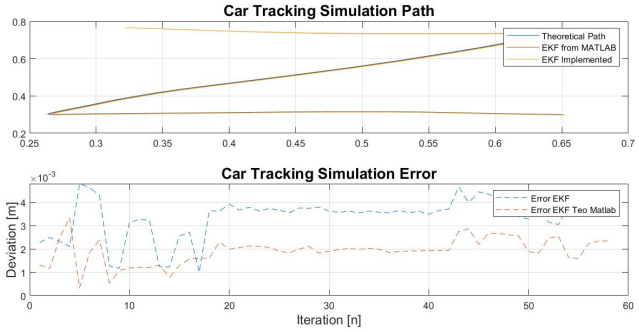


Fig. 25. *'Z'* letter with High Measurement Error Conditions

3) Test on Convergence Ability: In this final individual experiment, the objective is to study the convergence ability of our implementation by starting the filter with a different position than the actual car, which then through the measurements should, in theory, converge to the right path. As seen in Fig.26, the initial error is quite high since the model starts in a different initial position than the actual car, but at *n=20*, both filters start to converge to the actual car path, substantially reducing the error. This meaning, that both filters managed to convergence when the initial position isn't the same as the car, but after more rounds of testing and increasing the initial deviation, it was possible to conclude that the convergence ability has its limits, and to increase it, it would be needed better or even more sensors.

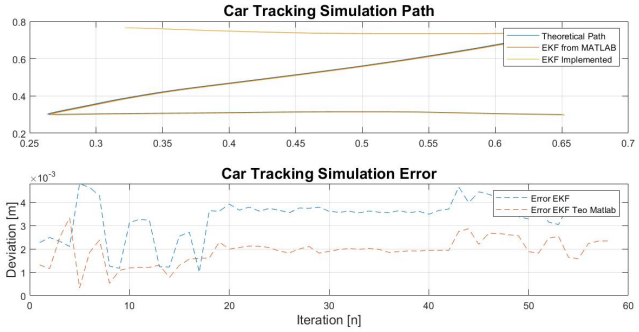


Fig. 26. *'Z'* letter with Different Initial Position

4) Simulation of the sensors: In the Figure below, Fig.27, we can see a pedestrian, represented by the red lozenge, that is detected by the camera sensor, represented in green. When the camera detects this pedestrian, a pop-up is displayed to inform the user of what is happening in the simulation.

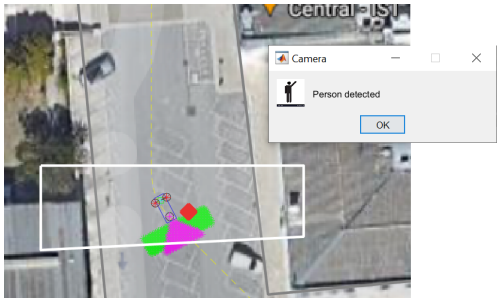


Fig. 27. Display of car detecting a person

In the Figure below, Fig. 28, we can see a pedestrian, represented by the red lozenge, colliding with the car. When this happens, the proximity sensors of the car detected the collision with the pedestrian, if the pedestrian is less than a threshold near the car. A pop-up is displayed to inform the user of that collision and increments the number of total collisions.

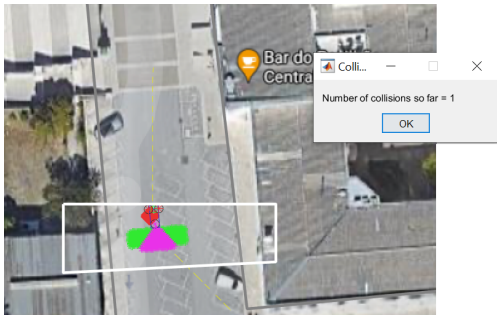


Fig. 28. Display of car colliding with a person

VI. Control

The Control block connects directly to the car dynamics model. The main purpose of the controller is to determine the linear and angular velocity that enables the car to follow the path given by guidance. In order to predict the car's position, the controller relies on the data given by navigation, since their sensors are fundamental for position and surrounding prediction. Overall, the control block has the task of following a path with the help of the other three blocks.

A. Control Principles

In control there are two principles that guide the ideal projection of a system. A system may be described as two main blocks: the simulator and the controller. The simulator should be as complete as possible, in order to be a faithful simulation of reality. This means that a good simulator should have all the non-linearities of a model, allowing a good analysis of the real system behaviour, with all possible noise and creating a good trial for the controllers.

While the simulator must be as close to reality, the controller should be as simplified as possible. Since most of the systems have non-linearities, the response of a controller that tries to control all oscillations tends to be slow and even not stable. Complex models are much more harder to manipulate and creating a controller for a specific problem it is not a good choice since it will not adapt to changes in the problem, which means that the projected controller can only control a very specific system in a very specific situation.

B. Car Simulator

Before trying to control the car movement, it is important to create a good simulator of the car in order to fully understand the car motion model.

For the purpose of this work, it will be consider the *simple car* model, which is composed by one front wheel and two back wheels as represented in Fig.29. The car dimensions are represented at Table IX.

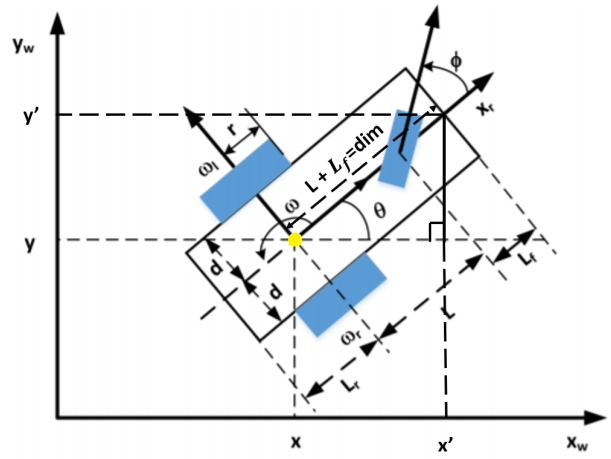


Fig. 29. Simple car model.

TABLE IX
Car dimensions and Specifications

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

A car moves in a three-dimension space, x, y, and θ, where x and y are represented by a yellow point in Fig.29. However, we cannot consider that the car can freely move in this space. Since the steering wheel (front wheel) is the only one that can be turned, the car cannot move sideways, making most of its trajectories obliques.

In order to accurately simulate a car environment, an initial configuration is needed. Let us consider a car with initial configuration $q_1 = (x_1, y_1, \theta_1)$. In Fig.30, there is a new representation of the car to explain the motion model. Let θ denote the angle of the car referent to the origin and Φ the steering angle. Let v denote the car's speed. There are two measures needed for the model, the distance between the front and rear axles, L, and ρ, that represents the radius of the circle made if the car had moved with a fixed Φ.

In order to predict the next state, let us consider the following equations.

$$\dot{x} = f_1(x, y, \theta, v, \Phi) \tag{23a}$$
$$\dot{y} = f_2(x, y, \theta, v, \Phi) \tag{23b}$$
$$\dot{\theta} = f_3(x, y, \theta, v, \Phi) \tag{23c}$$

Let us also consider a small time interval, Δt. We can easily see that, for each instant of time t, the car will move in the approximate direction

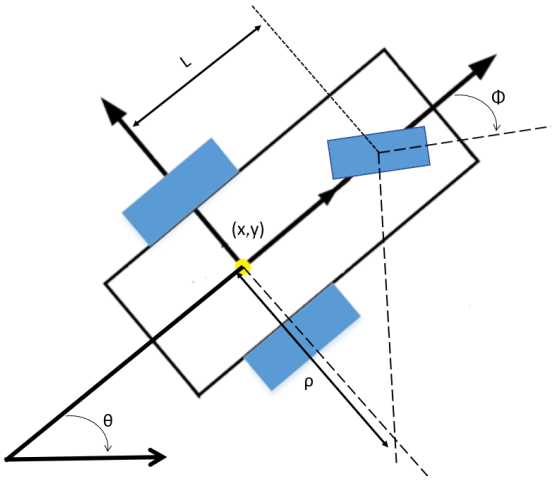


Fig. 30. Car configuration and important measurements for simulation block.

of the rear wheels. Accordingly, when Δt tends to zero, we can assume that

$$\frac{dy}{dx} = \tan\theta. \tag{24}$$

We also know that

$$\frac{dy}{dx} = \frac{\dot{y}}{\dot{x}} \tag{25a}$$

$$\tan\theta = \frac{\sin\theta}{\cos\theta}. \tag{25b}$$

With the expressions in (25), condition (24) can be written as a *Pfaffian* constraint[4].

$$-\dot{x}\sin\theta + \dot{y}\cos\theta = 0. \tag{26}$$

This constraint can be satisfied if $\dot{x} = k\cos\theta$ and $\dot{y} = k\sin\theta$, where k is a scaling factor. For the purpose of this problem, let us consider that $k = v$. Afterwards, it is necessary to derive the equation for θ. Let ω denote the distance travelled by the car. We can easily check that $d\omega = \rho d\theta$ and $\rho = L/\tan\Phi$, so it is possible to conclude that

$$d\theta = \frac{\tan\Phi}{L} d\omega \leftrightarrow \dot{\theta} = \frac{v}{L}\tan\theta, \tag{27}$$

since $d\omega$ is the integral of v. These equations resume the simple model for the car simulator. However, since is desired a simulator as closer as possible to reality, it is not possible to assume that the steering angle orientation can change instantaneously, so we should notice that $d\Phi = \omega_s$. Overall, the transition equations for the car are

$$\dot{x} = v\cos\theta \tag{28a}$$

$$\dot{y} = v\sin\theta \tag{28b}$$

$$\dot{\theta} = \frac{v}{L}\tan\Phi \tag{28c}$$

$$\dot{\Phi} = \omega_s. \tag{28d}$$

Once determined the transition equations for the car, it is possible to conclude that $q_{k+1} = (x_{k+1}, y_{k+1}, \theta_{k+1}) = (x_k, y_k, \theta_k) + (\dot{x}, \dot{y}, \dot{\theta}) \cdot \Delta t$ for small values of Δt. However, one should note that we are modelling a car, therefore, there are some limitations for the parameters v, ω_s and Φ. Since the car will move into a specific zone, it was decided to limit the velocity for 20 km/h ≈ 5.6 m/s. The car movement should also be smooth, so the steering velocity, ω_s, cannot be higher than 1.5 rad/s. Finally, let us consider that the car wheels belong to a specific range of angles. In this case, we can assume that the maximum turning angle is $\pi/4$ rad.

[4]A *Pfaffian* [11] constraint is a way to describe a dynamical system as a linear in the form $f_1(r)\dot{r_1} + f_2(r)\dot{r_2} + ... + f_n(r)\dot{r_n} = 0$. This is only possible under the assumption that r is known.

C. Energy consumption

ONE of the most important features in an electric car is the energy budget. The operation of the car results in a spend of energy, so it is necessary to keep track of this value in order to make a real simulation. When the energy reaches zero, the car systems stop working and the car naturally stops. In this situation there is no control over the car and the sensors will not work too.

In order to keep track of the car remaining energy, we should analyse the energy spent over time. For each instant of time, the energy spent by the car is given by

$$\Delta E = (|F(t)||v(t)| + P_0)\Delta t = (M|\dot{v}(t)||v(t)| + P_0)\Delta t, \tag{29}$$

where P_0 is the constant cost of having the car in operation but not moving[5].

The value of the velocity is the same as predicted in the controller. This control of the total energy of the car is not only used to know when the car is out of battery, but also in the navigation block, V, in order to estimate the maximum velocity that the car can reach to end the desired path without getting out of energy.

D. Control Signal

THE development of a controller, as previously mentioned, should aim for a simple structure that cannot be too much focused on the particularities of the path or the mission, in order to be robust to changes. Intending to accomplish this characteristic, the controller must focus on simple objectives during the mission.

The first important parameter that needs to be analysed is what should the controller follow. This particular low-level controller is developed to only follow the path given by the guidance block. However, the tracking problem can be interpreted as following a trajectory or following a static configuration.

For the first approach, it is desired that the car follows a given trajectory. Following a trajectory means that the car must achieve the goal of reaching a configuration q at time t, which means that after reaching that configuration, the car must be ready to keep on going to the subsequent configurations.

The second approach follows a more strict model. In order to follow a static configuration, let us assume that the car should converge for every single configuration of the path. For this approach, the car will only move into the next configuration if the previous one was been achieved.

To solve the problem, it was not necessary that the car converged to each of the path configurations. However, trying to follow a path trajectory was not the best alternative, since with that the car velocity would be dependent on the path. Due to these limitations, a new approach was developed.

Let us consider a perfect car that can perfectly follow the given path, at the velocity as the real car, with configuration $q^{perfect} = (x^{perfect}, y^{perfect}, \theta^{perfect})$. Considering that if the real car could overlap the perfect car, the task would be successful. Now, instead of following a path, the task should be to follow the perfect car. Since this car moves at the same speed of the real car, the following process would be much smoother and easier. An example of this approach is represented in Fig.31.

[5]For the purpose of this project, we decided to use $P_0 = 1000$ w.

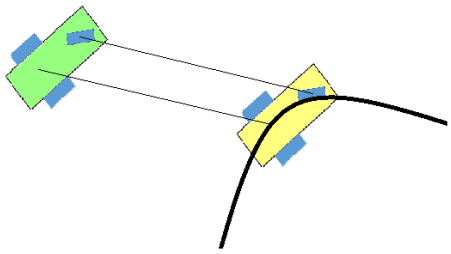


Fig. 31. Real car (green) following the perfect car (yellow).

In order to create this perfect car simulation, it is necessary to have all the configurations q for the perfect car and the position of the real car. From the guidance block is given a path to follow, with x and y, the initial orientation, θ_0, and the final orientation, θ_N. The orientation for each of the configurations can be easily obtained with an approximation by sections, where each section is the straight line that connects two positions. From this section, the orientation is given by $\theta_k = atan2(\dot{y}_k, \dot{x}_k)$[6]. The prediction of the car position is given by the estimation of the *EKF* from the navigation block.

With the perfect car simulation, the real car just needs to follow this simulation. However, trying to follow the closest car is not the best choice. The car movement must be as smooth as possible and, for that, the approach used was to follow the simulation a few steps ahead. In this step there is a parameter, stp, that needs to be optimised in order to get the best results for each road.

E. Controller

AFTER defining the model for the car and the control signal, it is now possible to project a controller for this model. The car model has two input parameters: the velocity of the car v and the steering wheel velocity ω_s, accordingly to (28).

In order to follow the control signal, we need to project v and ω_s. For each iteration, the controller will try to minimize the error between the position of the car and the position of the perfect simulated car. Since both positions are known, it is possible to determine an ideal θ that allows the perfect convergence. From (28a) and (28b) it is possible to conclude that

$$\theta^{ideal} = atan2(\frac{y^{perfect} - y^{real}}{v}, \frac{x^{perfect} - x^{real}}{v}). \quad (30)$$

However, as stated before, this approach would only be adequate if we were following a static configuration. Since a perfect car is being followed, the goal is to have the same orientation as that perfect car. Accordingly, there are two possible choices for the θ_k that it is aim to get, $\theta_k = \theta_k^{ideal}$ or $\theta_k = \theta_k^{perfect}$. Since both values are needed for the correct convergence, the best alternative is to do a mean value of the variation for each alternative

$$\dot{\theta} = \frac{(\theta_k^{ideal} - \theta_k) + (\theta_k^{perfect} - \theta_k)}{2}. \quad (31)$$

It is important to notice that the difference between orientation values is done accordingly to the unit circle[7].

[6]The function $atan2$, also known as *Four-Quadrant inverse tangent*, returns values in the interval $[-\pi, \pi]$. It is a programming function that tries to correct the ambiguity of $\arctan$.

[7]The difference between angles should be done using the unit circle. Since every angle belongs to $[-\pi, \pi]$, the max difference between angles is π.

With the desired $\dot{\theta}$, it is possible to calculate the ideal value for the wheel orientation,

$$\Phi_k^{ideal} = atan2(L \cdot \dot{\theta}, v). \quad (32)$$

At this point, the four transition values are expressed in (28). From those values, it possible to obtain the ideal values for the velocity of the car and the wheel turning speed. Without any of the events defined in V, it is possible to control the car within a few situations.

For the wheel turning speed, it is important to keep in mind the variation along the path. For that, it was used a simple proportional controller, where the desired value is given by

$$\omega_s = 10 * (\Phi_k^{ideal} - \Phi_k), \quad (33)$$

where the 10 is the proportional factor.

Another alternative for this controller was to use a PID[8] controller that keeps track not only of the instant variation but also of the cumulative variation of this value. However, after a few tests, it was noticed that the presence of the integrative and derivative terms make the model more unstable since ω_s is restricted to $[-1.5, 1.5]$, in order to make a smooth drive.

For the velocity of the car, the path complexity must be checked. However, this complexity should be checked locally since the velocity parameter is changed during the progress. The approach used consisted in developing a function that would look into the following ten meters of the path, and generate an orientation variation[9] for that segment. With that variation, it was decided the maximum speed of the car in order to be able to do the curves. The wheel orientation is also important when deciding the velocity of the car. After that, it was analysed the value of Φ^{ideal} and regulated the car's velocity, in a way that it would only accelerate if the wheel's angle is approximately zero[10].

With all these restrictions taken into account, the car velocity can be expressed by

$$v = \frac{10(x^{perfect} - x^{real})}{\cos \theta_k^{ideal}}, \quad (34)$$

where 10 is a proportional factor.

If the variation in x is approximately zero, v can be calculated using y

$$v = \frac{10(y^{perfect} - y^{real})}{\sin \theta_k^{ideal}}. \quad (35)$$

With the main parameters discovered, just a few modifications were implemented in order to get a more realistic approach. To make the drive more smooth, the wheel orientation was restricted by changing between its maximum and minimum value. For that, if $\Phi = \pi/4$ and, from (32), the desired value for the next iteration is $\Phi^{ideal} = -\pi/4$, the controller should reject that change and keep the same value. Alternatively, if the desired value for the next iteration is the maximum or the minimum and $\Phi^{ideal} - \Phi > \frac{5 \cdot \pi}{16}$, Φ^{ideal} should change to its negative value, $-\Phi^{ideal}$. The braking process is also an important feature in a car. With that being said, it was added a coefficient relative to the friction of the road in order for the simulated car to be as realistic as possible. A normal value for the coefficient of friction of a road is $\mu = 0.7$ [12]. Also, it is known that

[8]Proportional-Integral-Derivative controller.

[9]This variation is determined from the difference between the actual θ of the car and the mean θ of the path segment.

[10]It is important to notice that in this program, negative velocities were not implemented and the car's velocity is always restricted by the limit velocity chosen.

$$F = m \cdot a \tag{36a}$$
$$F^{frict} = -\mu \cdot m \cdot g \tag{36b}$$
$$a = \dot{v} = -\mu \cdot g. \tag{36c}$$

From (36) it is possible to check that the maximum variation for the velocity of the car while braking is given by (36c). With the braking process, an acceleration model was also used in order to make the whole process more realistic. After some research, it was decided that a good acceleration value would be $a = 3.3\ m/s^2$[11].

F. Odometry

THE odometry is a parameter directly related to the car movement. For that it was necessary to calculate this parameter with the data from the simulator. The odometry is calculated by having the motion of the wheels. It is important to notice that the odometry does not give a position for the car but the translation of the car between two iterations.

For the purpose of this project, let us assume that all the specifications defined in Table IX are not perfect, which means that there is some deviation from the perfect odometry, just like a real car. For that it was assumed that the odometry applies a standard deviation to the right.

The odometry's error is a cumulative error which means that, over time, the prediction of the car's position given by odometry gets worse. For that, in a way to simulate a wheel odometer, it is possible to apply

$$x_{k+1}^{odom} = x_k^{odom} + \epsilon \cdot \sin\theta + \dot{x} \tag{37a}$$
$$y_{k+1}^{odom} = y_k^{odom} + \epsilon \cdot \cos\theta + \dot{y}. \tag{37b}$$

In (37) ϵ is a constant cumulative error. Let $\dot{x}$ and $\dot{y}$ denote the same variables expressed in 28a and 28b. A simulation of the odometry prediction is represented in Fig.32.

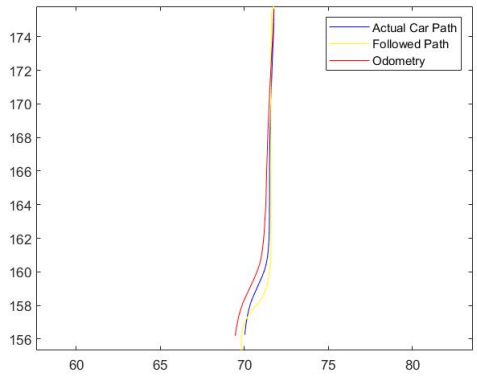


Fig. 32. Linear path with the true position (green) and the odometry prediction (red).

G. Parameters Regulation

ALONG the section VI it was defined a set of parameters that needed to be regulated in order to provide a good controller. The proportional parameters used in the velocity equations where chosen by trial and error, applied to simple paths in order to keep a more smooth drive. This parameters can easily be applied to any path in any map since they are adapted to the cars behaviour.

[11]Equivalent to 8.4s from 0 km to 100 km, which is an average value for a mid class vehicle.

A more challenging parameter was the value of stp introduced in the simulated perfect car. In order to follow this car in a more realistic way, the real car should focus on a few iterations ahead, since the objective is to converge to the future iterations. The number of iterations is dependent on this parameter. The effect of this parameter in the number of iterations ahead is

$$it_{jump} = \left\lceil \frac{1}{stp} \right\rceil .^{12} \tag{38}$$

This parameter is also discovered by a trial and error method. However, the value of stp is specific for each path, so it should be calculated in the beginning of the program. To get a good prediction of this parameter, we should simulate the whole process for a set of alternative values. From our tests, we discovered that the best values for stp are in the interval $[0.001, 0.1]$. Accordingly, the whole path was simulated for every value in that interval, with a minimum step of 0.001.

To decide which value is the best, it was analysed the error along the path and, with that, the value with the lower error is chosen. To make this approach faster, it was discarded values with errors bigger than a pre-specified threshold.

Since it is desired to this prediction to be as accurate as possible, the whole simulation is done with the predictions given by navigation.

H. Special Events

WHILE trying to estimate a good value for the velocity of the car and the wheel turning speed, the controller must also be alert to a few flags that indicates changes in its way. A list of the expected flags and their meaning can be viewed in table X.

TABLE X
CONTROL FLAGS

Wet	Wet floor
Stop	Stop the car
Slow Down	Reduce the car velocity
Person	The presence of a pedestrian in the road
End Stop	The end of the path is nearby

1) Wet Flag: The simpler flag, independent of the segment of the path. This flag indicates that the road is wet and, resulting in a lower friction between the car and the road. Because of this, the car stopping distance is increased meaning that the car cannot react as fast as before.

2) Stop Flag: The Stop flag is one of the most important ones. This flag is responsible for making the car stop when there is any kind of obstacle. At a first implementation, this flag was used to avoid collisions, however, since one of the objectives was to count the collisions, instead of stopping, the controller tries to minimize the velocity. At its final implementation, this flag only notifies about the presence of a red light, the presence of a stop sign and that the car has no more battery. All these stops are for a limited amount of time, except the one when the car has no more battery, which ends the whole simulation.

3) Slow Down Flag: This flag was implemented to make the car slow down in some areas or in some dangerous situations. As requested in the guide, the car must slow down in cross-walks and thus this flag is used for those situations. Besides slowing down in cross-walks, the car should also slow down when there is an inherent collision. In this case, the car slows down in order to avoid the collision, even though sometimes it is not possible.

4) Person Flag: This flag is pretty self explanatory. When the sensors detect a person, the car must stop in order to let the person move away from the front of the car. In some situations, the car would not be able to avoid the person since his velocity is very high. If the person is on a cross walk, since the car is slowed down, it is almost

[12]This symbol represents the ceil operation that always rounds up the fraction value.

certain that the car would stop. However, if the person is on the middle of a straight street, then the car is at his highest velocity and it may not be able to stop in time.

5) End Stop Flag: The final flag was implemented in order to facilitate the process of ending the path and enabling a more realistic scenario. In real life, the car cannot instantly stop at the end of the path so, in order to make this simulation realistic, when the car is close to the end, the controller starts to indicate a slow down process. Although this flag is very similar to the Slow Down Flag, the End Stop Flag distinguishes by its brake process, which is definite.

I. Control Simulation

THIS section has the purpose of showing some important tests that try to cover the controller main features. At first it is shown one test in a simple curve that it will be denominated as the control test, which is used to analyse the results of the other tests. The other test will cover some special situations that may effect the car controller efficiency, the presence of Gaussian noise in the prediction given by the navigation clock and a wet road.

1) Control Test: The control test was done near the Math's department building. The main purpose of this test is to give the expected results for this curve without any additional parameters. The path done by the car can be seen in Fig.33.

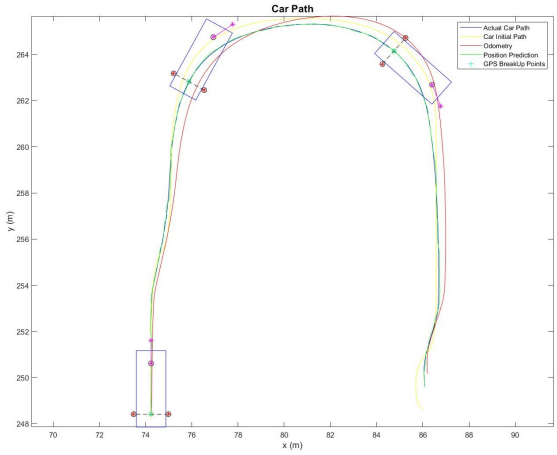


Fig. 33. Path chosen for the test.

In Fig.33 is possible to verify that the car is able to follow the path without too much deviation. However, in order to make it easier to evaluate the cars performance, the error and the velocity evolution are represented in Fig. 34 and Fig.35, respectively.

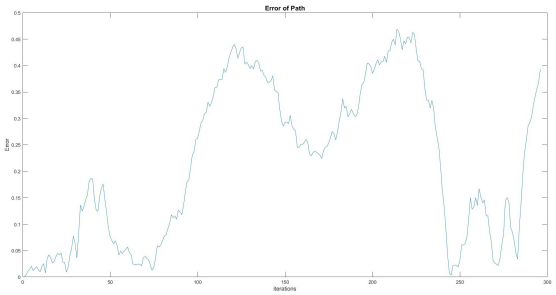


Fig. 34. Error in the control test.

For this control test, the error is always lower than 0.5 meters, which means that the car can follow the path without major deviations. Within the path presented at Fig.33, there are two segments in which we expect a biggest error: the two curves. From Fig.34 is possible to verify that the error is bigger in these two segments, being almost four times bigger than the mean error. However, there is another segment where the error is also bigger: the end of the path. In order to make the simulation more realistic, the car starts to slow down as it gets closer to the end. With that, the error between the car and the predicted position gets bigger. In this path, there is also a small curve in the end of the path, which contributes to this noise increase.

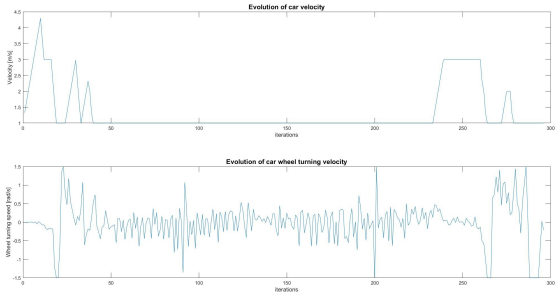


Fig. 35. Velocity evolution in the control test.

From Fig.35 is possible to verify that the car linear velocity increases in the beginning and in the end of the simulation, corresponding to the segments without curves. At the middle of the simulation, the controller tries to increase the velocity of the car, but since the curves are too close, it goes back to the minimum value. In the end, the velocity is also in its minimum value, as explained before. For the wheel turning speed, its value is only zero at the beginning, meaning that the wheel is only in the perfect position when the car starts aligned with the path. The biggest variations correspond to the curves segment, being the moment where the car needs to change its orientation.

2) Gaussian Noise Test: The presence of Gaussian noise will make the converge of the car harder because the controller will never truly know where the car really is. In this test, we can preview an augmented error during the simulation, more variation in the wheel turning speed and a lower linear velocity. These results can be observed in the Fig.36 and Fig.37.

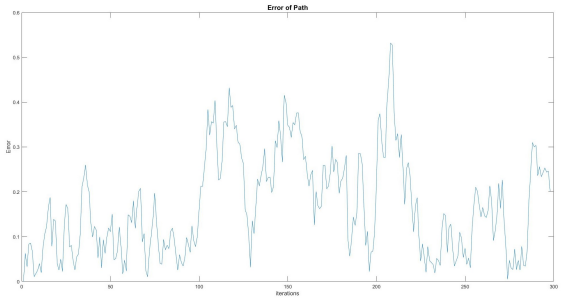


Fig. 36. Error with Gaussian noise in the location prediction.

After a brief analysis of the results, it is possible to conclude that not only the error was bigger, it was more inconsistent, being this an effect of the presence of Gaussian noise. While the wheel turning speed behaviour was the expected one, with more variations, the car linear velocity did not behave as expected, since its evolution was almost equal to the one in the control test.

3) Wet floor Test: When the floor is wet, the coefficient of friction decreases to approximately 0.5, which means that the car needs more iterations to reduce the velocity. This effect can increase the error in the places where the car should brake, but will also make the velocity variation more smooth. The results of this test are presented in Fig.38 and Fig.39.

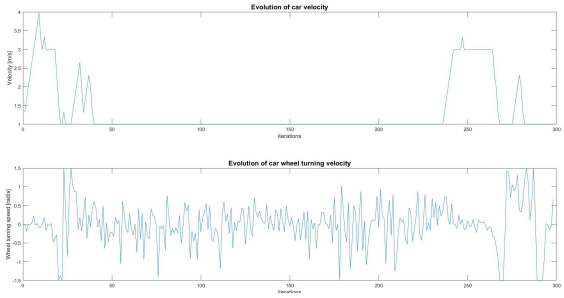


Fig. 37. Velocity evolution with Gaussian noise in the location prediction.

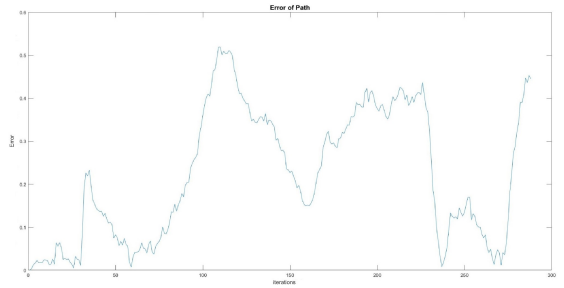


Fig. 38. Error with wet floor.

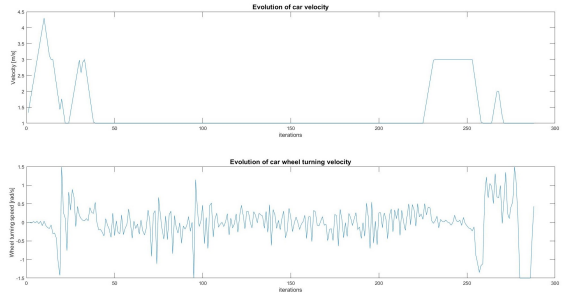


Fig. 39. Velocity evolution with wet floor.

In this case, the error is actually bigger than without wet environment but its smaller when compared to the presence of Gaussian noise. As expected, the linear velocity evolution was more smooth compared to the other situations. In the previous situations, the velocity of the car had some peaks that were mitigated in this simulation. The wheel turning speed was not affected by this parameter.

4) Hard Test: As a final simulation, the controller behaviour was tested in a track that would be very hard to follow. In this track, the car must do a very closed curve, corresponding to the event of *making a u-turn*. This is one of the most challenging driving manoeuvres and the main purpose of this test is to verify if the controller can execute this task. The path intended to follow and the car response are represented in the Fig.40.

In Fig.40, is easy to spot a deviation of the car trajectory from the predefined path. However, this deviation does not implies error in the subsequent segments of the path which means that the car is able to recover from that deviation. Since the controller predicts the future stages, the choice of following this trajectory was done in order to minimize the error in a curve that the car would not be able to do perfectly. The presence of GPS breakups, as explained in section V, makes this turn even harder since the position prediction is not as accurate. This test was also done with Gaussian noise associated to that prediction.

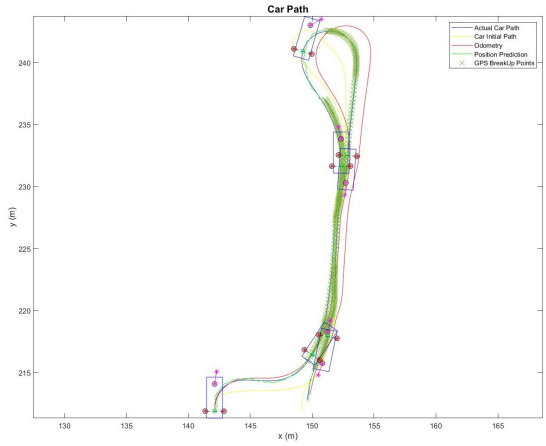


Fig. 40. *U-turn* path.

In order to make it easier to evaluate the simulation, the error is presented in Fig.41 and the velocity evolution is presented in Fig.42. In Fig.41, is possible to verify that the error increases in the begging of the path to more than one meter. This amount of error is almost bigger than the one in the *U-turn* curve. The prediction step, stp, is optimized in order to minimize the error along the path, which means that when there are two curves that would require different values for the stp, the algorithm tries to find a value that minimizes the overall error, which means that it would choose a step that works poorly in some easier curves just to lower the error in the harder curve. At the end of the simulation, the error decreases to less than 0.6 meters, which means that the car is able to converge in the end.

According to Fig.42, is possible to verify that most of the time the cars velocity is at his minimum value, only increasing for brief moments. Is important to notice that since the results are show in relation to the number of iterations, even though it seems that the car makes most of the path with low velocity, this is not true, since with higher velocity the car travels a bigger distance in less iterations. The variation of the wheel speed is also interesting to analyse, since most of the way it varies a lot. However, there is a segment where this speed is zero, representing the moment of the *U-turn*. At this moment, the car turning angle must be stable in order to make the curve, so the wheel turning speed must be zero or the curve cannot be done successfully.

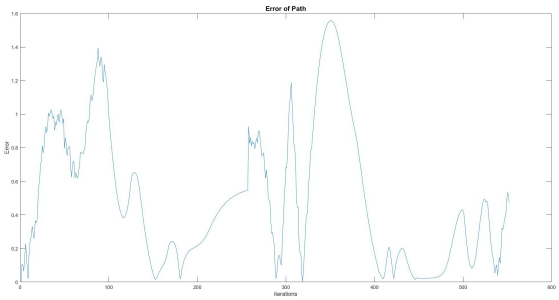


Fig. 41. Error in the hard test.

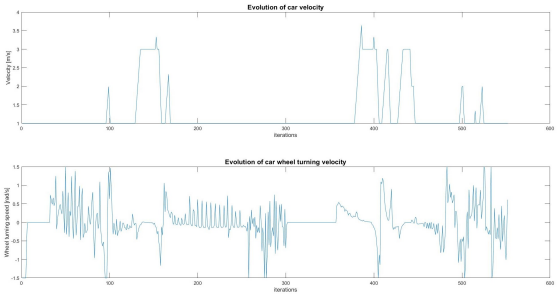


Fig. 42. Velocity evolution in the hard test.

VII. Experimental Results

In this section, it will be shown the tests that were made in the oral presentation, the first, to test the GPS breakups and people traffic events and also the second to test the traffic lights and speed limit zones.

A. Scenario 1

In this test, it will be tested the behaviour of the car in a simple environment, where the start point is the white diamond located in front of the "Pavilhão de Informática I/II" and the end point the diamond in the left of "Infantário". The initial and final orientation given by the user were 180° and 90°, respectively. Moreover, it will also be tested the pre-defined GPS Breakups zones (yellow rectangles marked in the map in Fig. 43), where they were specifically placed to be in accordance with real life scenario of the GPS communication obstruction caused by both IST towers.

In Fig.43, it is represented the path computed to be used as reference for the car. It can be seen that the curves are not abrupt and so the given path looks realistic. Furthermore, it is important to notice that the car makes little adjustments to be in the center of the road and also that the car prefers to be in wider roads, in order to avoid collisions (it can be seen when the car passes next to "Bar do Pavilhão Central" and it chooses the wider road).

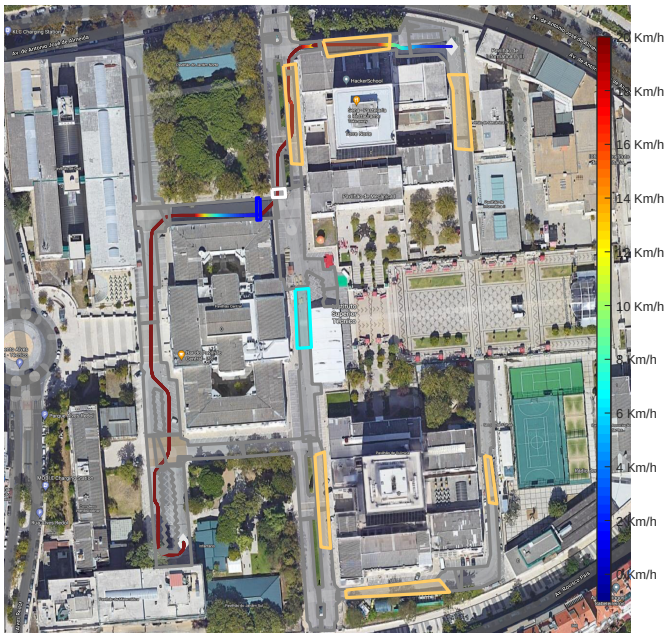


Fig. 43. Path given by Guidance.

In Table XI, it is represented the parameters predicted by the path planning, where one can notice that the error in orientation is not that significant. But also it is good to salient that this velocity is computed if there is no events throughout the course, that is, it does not take into account the crosswalks, traffic lights, etc. The cost along the path is represented in Fig. 44, that is, the cost of the Dijkstra's algorithm throughout time.

TABLE XI
Predicted values for the path planned

Distance [m]	392.42
Duration [s]	80.95
Mean Velocity [Km/h]	17.45
Speed [Km/h]	12.05
Initial Orientation Error[°]	0
Final Orientation Error[°]	2.80

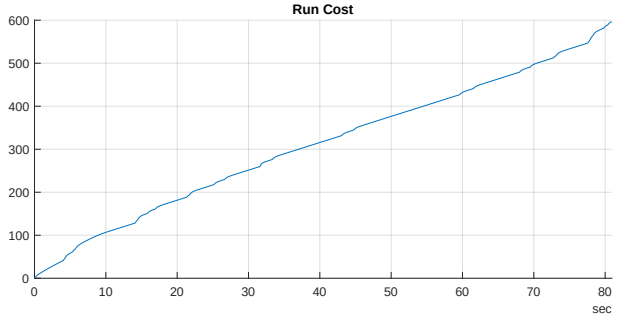


Fig. 44. Cost throughout the path.

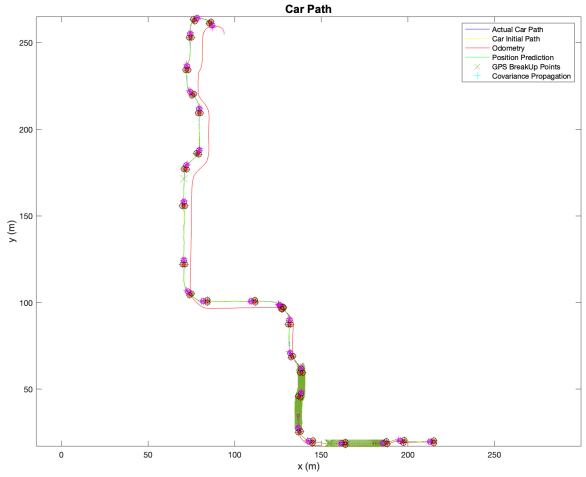


Fig. 45. Paths Generated By Guidance vs Actual path vs Path Estimate.

As it was previously mentioned in the theoretical part, an error was added to the outputs at the end of the *EKF* to replicate better a real-life scenario, and as expected the error at the end of the simulation showed in Fig. 46, has a lot of oscillations that come directly from this phenomenon. At some points the error remains constant due to a car stop. Since the car is stopped, the prediction remains the same, so does the error. This phenomenon can be spotted at the velocity evolution

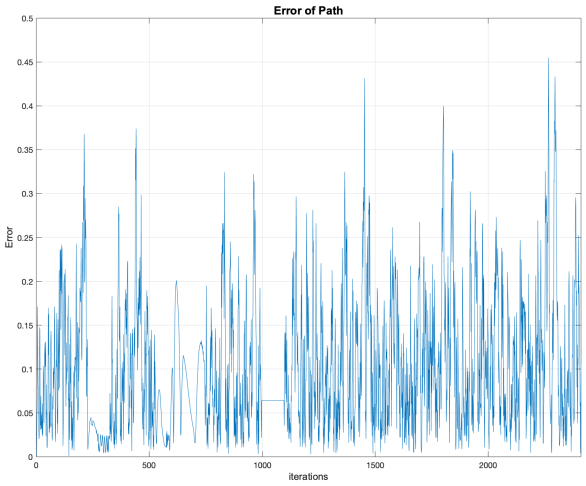


Fig. 46. Path Estimate Error.

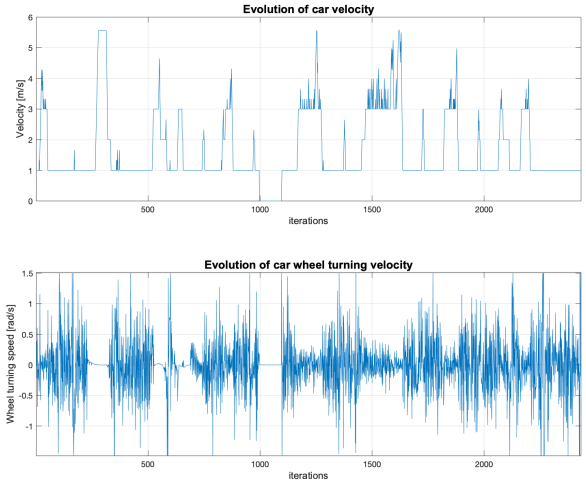


Fig. 47. Velocity evolution.

represented in Fig.47. It is important to notice that the error is bigger at the iterations corresponding to the curves.

Analysing the evolution of the velocities, is possible to see some interesting relations. At the iterations corresponding to the hardest curves, not only the linear velocity is at its lower bound, but the wheel turning speed is oscillating more. As previously explained in section VI, when the curves are too hard, the wheel turning speed tends to remain stable, so we can conclude that the curves of this path are acceptable. Is also possible to see that in the moments where the car is stopped, the controller does not try to move the wheel, which indicates a direct relation between the two velocities.

However, to illustrate that this is in fact true, it is presented the Fig.48 that showcases on the top image a zoomed GPS BreakUp Zone with both the path and covariance overlapping as spheres. From here, it is possible to understand that the covariance increases in the GPS breakup Zones since the sphere gets larger than the normal minimal spheres seen before. In addition, from the bottom image, it is possible to see both the output

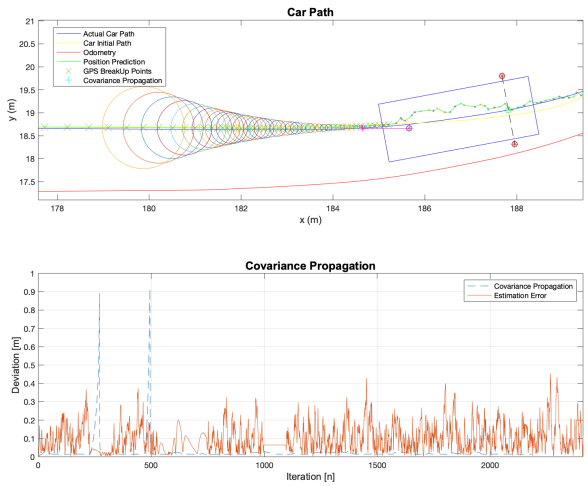


Fig. 48. Sensor Fusing - Covariance Propagation.

error and the covariance propagation, where the last one maintains a low error with few oscillations apart from the GPS breakup Zones and the other has much larger oscillations, which proves that these come from the added Gaussian error.

Since our main error came from GPS BreakUp Zones, an experiment was made in order to see if this error could be improved by instead of just propagating the Prediction Model, substitute in these scenarios the normal GPS Sensor by the Position Information from the Odometry, thus, in theory we substitute the GPS Sensor by the much Noisier Position measurement that comes from the Odometer. The results from the simulation can be seen in Fig. 49, and we can conclude that it actually made the errors worse than before, even probing the car to diverge due to the error accumulation i, which can be explained by the high error associated with the Odometry Measurements in comparison to GPS or by the sudden change in the sensor error (GPS $\rightarrow$ Odometer) which can induce high errors in the Covariance. Nevertheless, we can see from the image, that through this, the covariance remains low, while the error of the estimate increased drastically due to the deviation between the estimate and the Path.

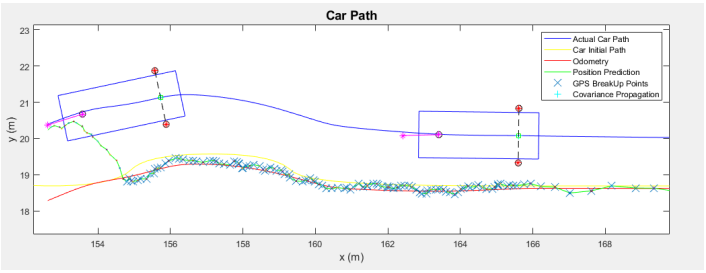


Fig. 49. GPS BreakUp Zone with Odometer as Sensor.

B. Scenario 2

In this test it is tested the effects of traffic lights, speed limit zones and STOP Signs, which are represented by the green rectangle, cyan rectangle and red polygon, respectively, in Fig.50. The initial and final orientation given by the user were 180° and 60°, respectively, and the

start point is the white diamond on top and the end point the white diamond on bottom, with a mid checkpoint in between them.

In Fig.50, it is represented the computed path to be used as reference for the car. It can be seen that the car has a circuit with wider roads than in the test 1, and so it does not need to be always readjusting the path, because it has no risk of collisions.

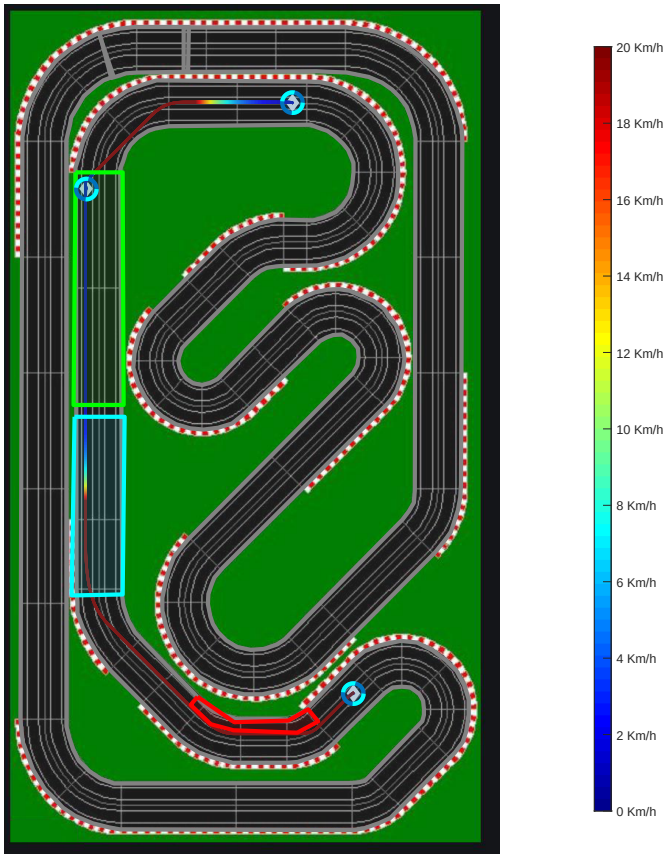


Fig. 50. Path given by Guidance.

The predicted values regardless the events and traffic lights are represented in Table XII. Due to the fact that path Dijkstra either begins or finishes with orientation of 0, 90, 180 or 270° before it is smoothed, it is expected that the error in the final orientation is bigger, because the final orientation asked by the user was 60°. The cost along the path is represented in Fig.51, that is, the cost of the Dijkstra's algorithm throughout time.

TABLE XII
PREDICTED VALUES FOR THE PATH PLANNED

Distance [m]	309.92
Duration [s]	87.38
Mean Velocity [Km/h]	12.77
Speed [Km/h]	7.65
Initial Orientation Error[°]	4.47
Final Orientation Error[°]	18.47

As seen in the previous example, it is possible to see from Fig.54 that the covariance associated with the *EKF* maintained low error values throughout the experiment, with the overall oscillation directly related with the gaussian error added to the final estimate to make the experiment a better simulation of the real environment. In addition, in this case, since there are no predefined regions for GPS Breakups, it was implemented the functionality of random zones which can clearly be seen by the sudden increase in both errors.

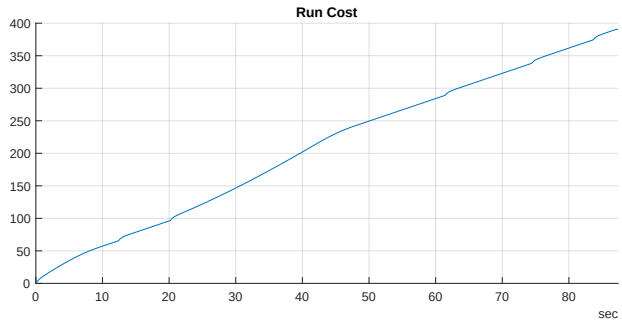


Fig. 51. Cost throughout the path.

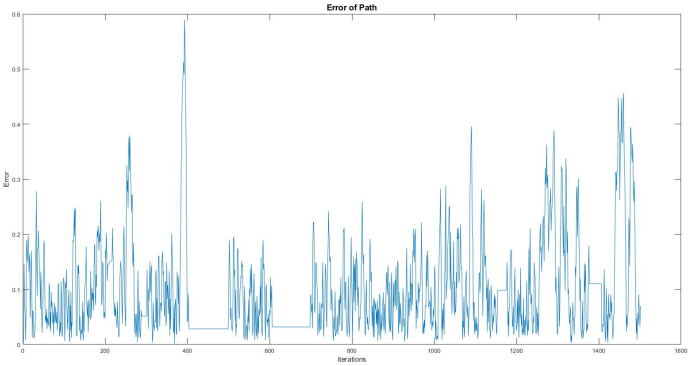


Fig. 52. Path Estimate Error.

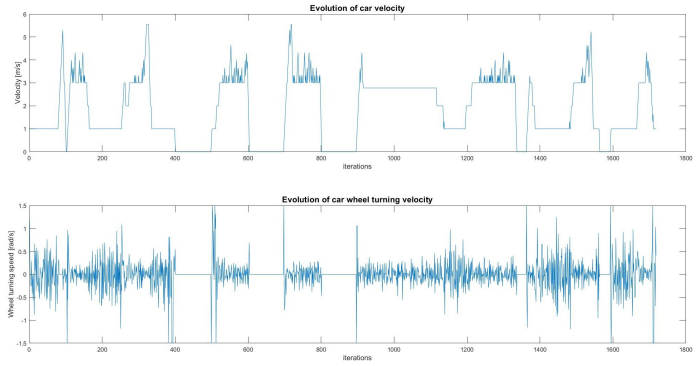


Fig. 53. Velocity evolution.

At Fig.52 there is a point where the error is bigger corresponding to the beginning of the traffic lights section, explained by the need to make a more closed curve. Overall the error is bellow 0.5 so we can conclude that it is a good estimate. In this example, from Fig.53, the linear velocity is almost every time at his higher bounds, since this map has less curves and the car can be at his maximum velocity. The wheel turning speed along the path is oscillating around low values, which is congruent with the high values of the linear velocity. Just like in the other example, when the car stops, both velocities are constant at zero.

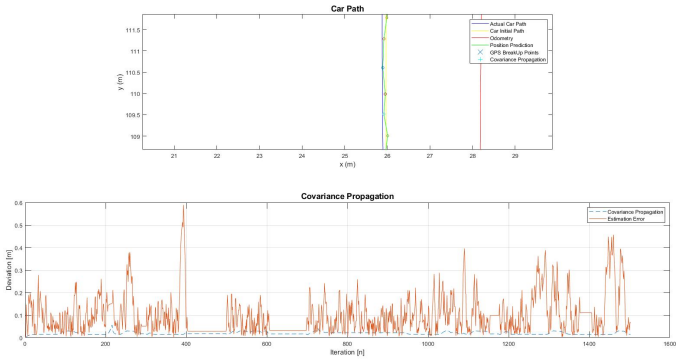


Fig. 54. Sensor Fusing - Covariance Propagation.

VIII. Conclusion

The main purpose of this project was to develop an autonomous car environment simulator. The main job was divided into three main blocks, a guidance block, a navigation block and a control block. The overall problem was solved with a combination of this three main blocks in order to enable a good behaviour for the autonomous car.

The guidance block develops a path given the street configuration and a few points in the path. This path was developed in order to maintain a safe distance from the dangerous zones, like out of the street zones. The path is created taking into account curves and streets that would be easier for the car to follow.

The navigation block tries to connect the sensors of the car in order to develop an estimate of the car position and surroundings. This fusion of data is done using information from the a set of sensors, GPS, odometer, camera, lidar, and proximity sensor. The information analysed by this block is of major importance for the car behaviour, since it affects not only the speed of the car, but also is ability to follow the path.

Finally the control block is responsible for following the path given by the guidance, with the information given by the navigation. Overall the controller is blind without any of the others blocks, being dependent of the remaining blocks. The ability to follow a given path is dependent of the difficulty of the path and the information read by the sensors.

The fusion of the three blocks was of major importance for this project, since a good cooperation means good results. Given the results analysed in section VII, is possible to verify that the combination of the three blocks was successful since the errors were mainly low. The overall detection of the random events were also successful, and with the connection with the controller, the car was able to follow the path without major incidents.

IX. Future Work

On the software demonstration, the professor rightly showcased a few modifications that could be made to the current implementation, mostly related to the display of the Traffic lights in the map, since there wasn't a clear distinction between the red and green lights when the car was reaching them.

In the guidance component, more improvements could be made regarding the cost function, for example, taking into account the altitude, the energy consumption or the traffic signs on a given road which could affect drastically the best path.

In the control component, more improvements could be made regarding the adaption of the controller to the path to follow. At this implementation, the controller tries to adapt to the whole path, which results into not having the best parameters for each segment of the path. A good improvement would be to consider every segment with different parameters, in order to minimize the error of each segment.

Finally, in the Navigation segment, a better version of the standard *Extended Kalman Filter* could be used to deal better with a Car Dynamic System, for example, the *Unscented Kalman Filter*, that would improve the nonlinear approximation, or even a *Particle Filter*, to improve the the estimate for a previously unknown environment, but only if the car processor would allow it.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

REFERENCES

[1] Tamatjita, Elizabeth & Mahastama, Aditya. (2016). Shortest Path with Dynamic Weight Implementation using Dijkstra's Algorithm. ComTech: Computer, Mathematics and Engineering Applications. 7. 161. 10.21512/comtech.v7i3.2534.

[2] Mathworks *Plot line in geographic coordinates* - https://www.mathworks.com/help/matlab/ref/geoplot.html

[3] MIT (2009) https://web.mit.edu/hyperbook/Patrikalakis-Maekawa-Cho/node17.html

[4] Ribeiro, Maria & Ribeiro, Isabel. (2004). *Kalman and Extended Kalman Filters: Concept, Derivation and Properties*.

[5] Kálmán. R.E. (1960) *"A New Approach to Linear Filtering and Prediction Problems"*.

[6] R. Hostettler, W. Birk and M. L. Nordenvaad, *"Extended Kalman filter for vehicle tracking using road surface vibration measurements,"*, 2012 IEEE 51st IEEE Conference on Decision and Control (CDC), 2012, pp. 5643-5648,

[7] Liang Zhao and C. Thorpe, "Qualitative and quantitative car tracking from a range image sequence," Proceedings. 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No.98CB36231), 1998, pp. 496-501, doi: 10.1109/CVPR.1998.698651.

[8] Yassine Zein, Mohamad Darwiche, Ossama Mokhiamar, *GPS tracking system for autonomous vehicles*,Alexandria Engineering Journal,Volume 57, Issue 4,2018,

[9] Hu, C, Wang, Z, Taghavifar, H et al. (2019) *MME-EKF-BasedPath-Tracking Control of Autonomous Vehicles Considering Input Saturation*. IEEE Transactions on Vehicular Technology, 68 (6). pp. 5246-5259. ISSN 0018-9545.

[10] MathWorks, *trackingEKF*, https://www.mathworks.com/help/fusion/ref/trackingekf.html

[11] Planning Algorithms, *Conversion from implicit to parametric form*, http://planning.cs.uiuc.edu/node656.html#eqn:pfaffian

[12] hyperphysics, *Friction and Automobile Tires* http://hyperphysics.phy-astr.gsu.edu/hbase/Mechanics/frictire.html

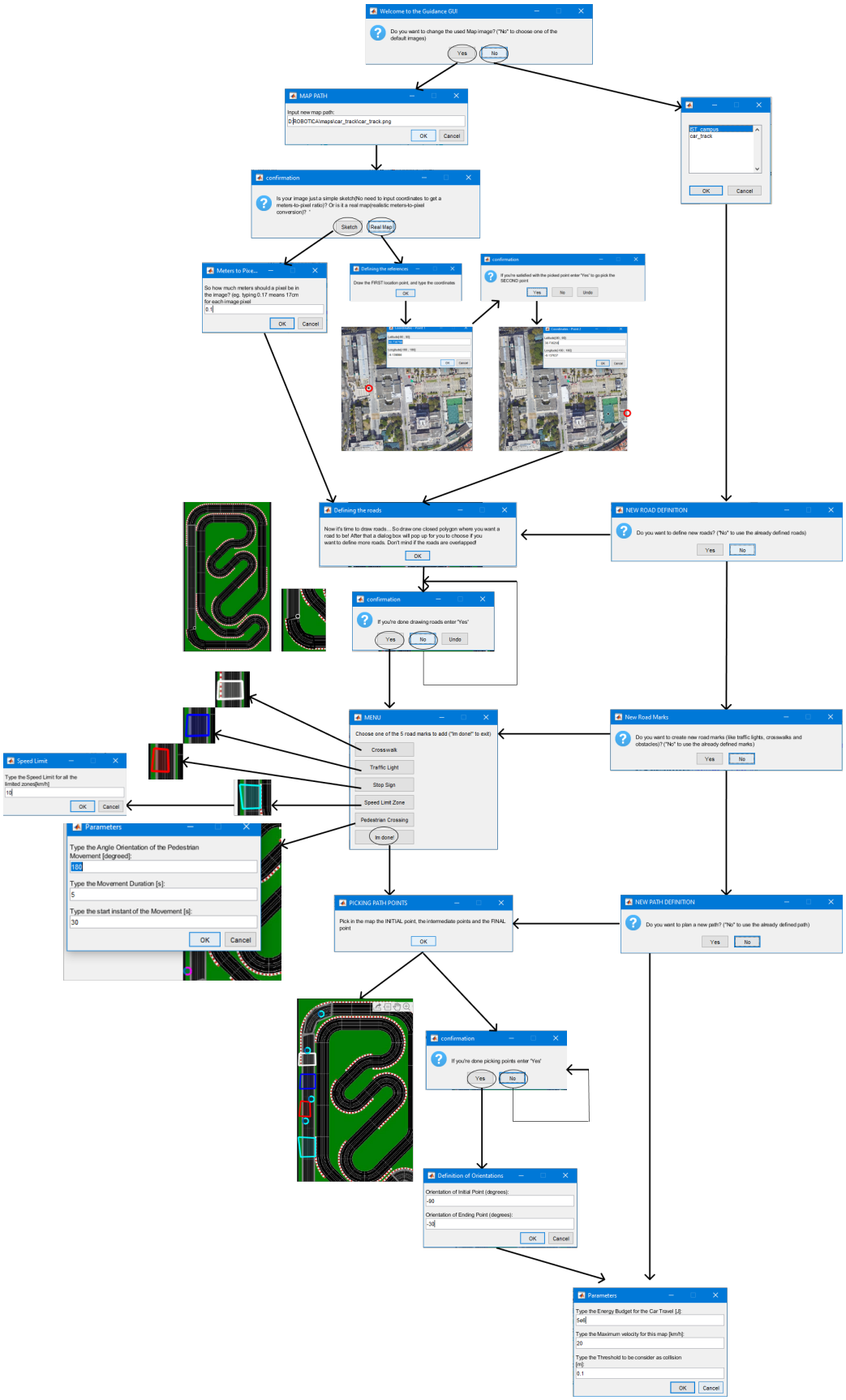


Fig. 55. Appendix 1 -Flow Chart tutorial for Software run