DEEC
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

Instituto Superior Técnico
Universidade de Lisboa

Robotics

2nd Quarter 2021/22

Diogo Antunes
93042

José Pedro
93109

Pedro Taborda
93152

October 13, 2022

Introduction

In this work the several components necessary for the development of an autonomous vehicle are shown.

A kinematic and dynamic model of an approximation of a car are determined, with which the autonomous navigation system can be verified.

The task of making an autonomous car is divided into three major components. First, there is the path planner, which takes information about the starting position, the destination and the map data, and generates the shortest path that the vehicle can take. In second place, there is the trajectory generator, which takes the path generated by the path planner and transforms the ordered set of waypoints into a trajectory, taking into account the available energy budget and the car's parameters. Finally, the controller takes the trajectory generated and tries to make the car follow the trajectory as best as possible.

The work developed shows promise, fulfilling the key points of an autonomous vehicle, being able to, given a destination, find a path and drive there, keeping itself on the road and maneuvering turns safely.

Further improvements to this system are discussed in the Section Further Work.

Car Model

The car model is divided into two components: the kinematics and the dynamics. The kinematics are described first, then the dynamics are determined with Lagrangian mechanics.

The model used is a planar simplification of a tricycle, described in the diagram in Figure 1.

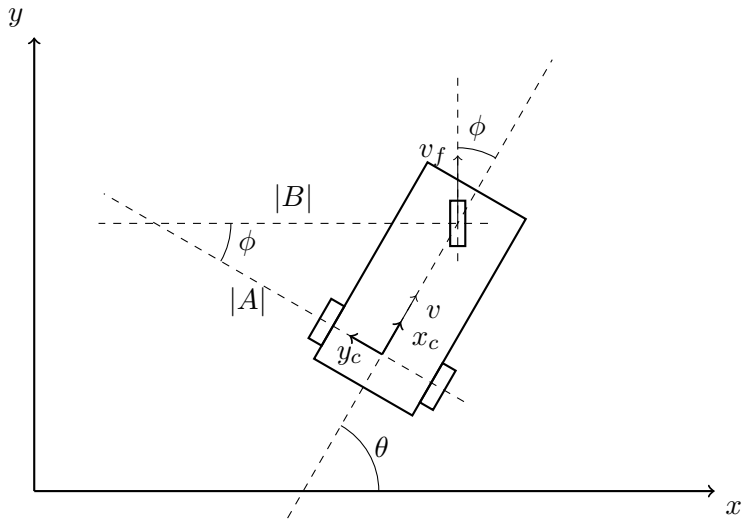

Figure 1: Diagram of the car model.

The 2D simplification is a close approximation of the real dynamics on flat ground and during normal operation.

The use of a single wheel in the front simplifies the derivation of the model and should not impact the results. This is because it is equivalent to a car with 4 wheels in which the 2 front wheels have slightly different orientations, but with the same overall movement. The problem of modeling the 2 wheels and designing a steering system such that the car would behave equivalently is a topic that can be dealt with separately and is not approached in this report. Consequently, the front wheel's steering is assumed to be controlled by some steering system that takes a steering angular velocity as an input and makes the wheels turn accordingly.

A model of the dynamics of the car's velocity is necessary, while the dynamics of the steering velocity can be safely ignored. This is because of the different relations between the forces required and the forces available in each case.

The energy cost of accelerating the car dominates the energy cost of the total system and the forces supplied by the car engine and breaks are not enough to make the car move as desired. As a result, the accelerations are slower than ideal and the time required to stop the car by breaking is longer than ideal, which can result in collisions with obstacles. For these reasons, a dynamic model of the car's velocity is essential.

The opposite is true in the case of the steering velocity. The energy costs of moving the steering wheel are comparatively irrelevant and the actuators to make it accelerate and stop as desired are available. For these reasons, a dynamic model of the steering wheel's velocity is not essential. Consequently, the model used for the steering wheel is purely kinematic.

Kinematics

The angular velocity of the car around its center of mass, ω, is the derivative of its heading, θ. As the car is a rigid body, the angular velocity is constant for every point on it. Assuming that the wheels don't slip, that is, their linear velocity is always perpendicular to their axis of rotation, the radius of the instantaneous curve followed by each wheel is perpendicular to the wheel. At the intersection of these radii is the instantaneous center of rotation. Thus, these radii are

$$A = \frac{L}{\tan\phi}, \qquad B = \frac{L}{\sin\phi}.$$

Since the angular velocity is constant, the relationship between v_f and v is given by

$$\omega = \omega \Leftrightarrow \frac{v_f}{B} = \frac{v}{A} \Leftrightarrow v = v_f \cos\phi.$$

Denoting the origin of the car frame as (x, y), the kinematic model of the car is

$$\begin{aligned} \dot{\theta} &= \omega = v_f \frac{\sin\phi}{L} \\ \dot{x} &= v\cos\theta = v_f \cos\phi\cos\theta \\ \dot{y} &= v\sin\theta = v_f \cos\phi\sin\theta. \end{aligned} \tag{1}$$

Dynamics

For the dynamics of the car, a Lagrangian formulation is used. Since there is an assumption about the world being 2D, with no height changes, the gravitational potential energy is constant, which for convenience was chosen to be 0. Considering no other potential energy contributions, the car has only kinetic energy.

No assumptions are made about the car's mass distribution in the determination of the model. Generically, a rigid body's dynamics in 2D are characterized by two parameters: its total mass, M, and its moment of inertia around the z axis, I_{zz}. This way, the kinetic energy of the car is

$$K = \frac{1}{2} M v_{CM}^2 + \frac{1}{2} I_{zz} \omega^2, \tag{2}$$

where v_{CM} is the velocity of the center of mass.

Since it is assumed that the car has front wheel drive, like most cars, it is desirable to express the energy as a function of the front wheel's velocity. For that, a relationship between the center of mass's velocity and the front wheel's must be obtained. In Figure 2, the geometric relationships necessary for this determination are shown for a generic position of the center of mass.

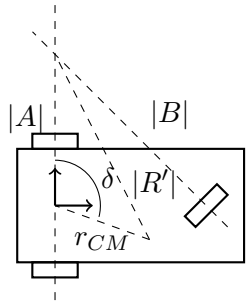

Figure 2: Generic center of mass position.

Denoting the position of the center of mass in the car frame as (x_{CM}, y_{CM}) and analyzing the figure it can be seen that $\delta = \text{atan2}(x_{CM}, y_{CM})$ and $r_{CM} = \sqrt{x_{CM}^2 + y_{CM}^2}$. Thus, applying the law of cosines, R' is given by

$$|R'| = \sqrt{A^2 + r_{CM}^2 - 2Ar_{CM}\cos\delta}.$$

Since the angular velocity is the same, the relationship between velocities is

$$v_{CM} = \frac{R'}{B}v_f = \sqrt{\cos^2\phi + r_{CM}'^2\sin^2\phi - r'_{CM}\sin 2\phi\cos\delta}\; v_f, \tag{3}$$

where $r'_{CM} = \frac{r_{CM}}{L}$. Thus substituting into (2),

$$K = \frac{1}{2}M(\cos^2\phi + r_{CM}'^2\sin^2\phi - r'_{CM}\sin 2\phi\cos\delta)v_f^2 + \frac{1}{2}\frac{I_{zz}}{L^2}\sin^2\phi v_f^2. \tag{4}$$

Since the potential energy is 0, the Lagrangian is simply $\mathcal{L} = K$. The generalized coordinates of this model are ϕ and s_f, which is the distance covered by the front wheel, such that $\dot{s}_f = v_f$. The necessary terms to compute the Euler-Lagrange equations are given by

$$\begin{aligned}
\frac{\partial\mathcal{L}}{\partial\phi} =& \frac{1}{2}\left[(r_{CM}'^2M + \frac{I_{zz}}{L^2} - M)\sin 2\phi - 2r'_{CM}\cos 2\phi\cos\delta\right]v_f^2 \\
\frac{d}{dt}\frac{\partial\mathcal{L}}{\partial\dot{\phi}} =& 0 \\
\frac{\partial\mathcal{L}}{\partial s_f} =& 0 \\
\frac{d}{dt}\frac{\partial\mathcal{L}}{\partial v_f} =& \left[M(\cos^2\phi + r_{CM}'^2\sin^2\phi - r'_{CM}\sin 2\phi\cos\delta) + \frac{I_{zz}}{L^2}\sin^2\phi\right]\dot{v}_f + \\
& \left[(r_{CM}'^2M + \frac{I_{zz}}{L^2} - M)\sin 2\phi - 2r'_{CM}\cos 2\phi\cos\delta\right]\dot{\phi}v_f
\end{aligned}$$

Thus, including the generalized forces associated with each generalized coordinate leads to the final model

$$\begin{aligned}
\tau_\phi &= -\frac{1}{2}\left[(r_{CM}'^2M + \frac{I_{zz}}{L^2} - M)\sin 2\phi - 2r'_{CM}\cos 2\phi\cos\delta\right]v_f^2 \\
\dot{v}_f &= \frac{F_v - \left[(r_{CM}'^2M + \frac{I_{zz}}{L^2} - M)\sin 2\phi - 2r'_{CM}\cos 2\phi\cos\delta\right]\dot{\phi}v_f}{M(\cos^2\phi + r_{CM}'^2\sin^2\phi - r'_{CM}\sin 2\phi\cos\delta) + \frac{I_{zz}}{L^2}\sin^2\phi}.
\end{aligned} \tag{5}$$

It should be noticed that the torque associated with the steering wheel angle is a static torque. The interpretation attributed to it is that it is the torque that has to be exerted on the steering

wheel's shaft in order for it to turn with the car. Since it is static, it is not included in the dynamic or kinematic model, under the assumption that this condition can and will always be satisfied.

To better mimic a real car, the steering angle is also limited. Also, as seen in (5), the front wheel acceleration depends on the steering wheel velocity, which means that this value cannot be infinite. Thus, the control variable for the steering wheel is the steering wheel velocity and not the angle. Taking all this into account, the complete model for the car are equations (1), (3), (5) and

$$\begin{matrix} -\frac{\pi}{3} \leq \phi \leq \frac{\pi}{3} \\ \dot{\phi} = \omega_s \end{matrix} . \tag{6}$$

Mass Distribution

A simple, yet powerful, mass distribution framework is to divide the car's mass into $n \times m$ uniformly spaced point masses, all with an equal fraction of the total mass. An example of this is shown in Figure 3.

For any distribution of mass composed of a finite number of point masses with positions $\boldsymbol{p}_i$, the center of mass is determined by

$$\boldsymbol{p}_{CM} = \frac{\sum_{i=1}^{m \times n} m_i \boldsymbol{p}_i}{\sum_{i=1}^{m \times n} m_i}.$$

For such a distribution, its moment of inertia is given by

$$I_{CM} = \sum_{i=1}^{m \times n} m_i ||\boldsymbol{p}_i||^2.$$

Throughout the rest of this work and as default in the simulation, the car's center of mass and moment of inertia are those determined for the distribution of mass shown in Figure 3.

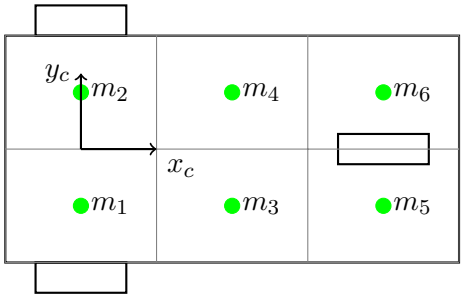

Figure 3: An example of a 3×2 mass distribution.

Path Planning

Map of the Area

The map of the simulation environment is fetched through the Google Maps Static API, which returns a limited resolution image representing a rectangular chunk of land centered on a point defined by latitude and longitude. Using this API it is possible to define map styles which eliminate unwanted elements, such as landscape features, markers for important buildings and street names. For this use case, the map style is defined such that the roads are displayed and everything else is hidden. Figure 4 exemplifies with an image of the roads at Instituto Superior Técnico.

Figure 4: Example of a map centered at Instituto Superior Técnico.

Road Resolution

The maximum resolution allowed for a given image for an API request is `640×640`px. This is small for simulation purposes. A zoomed-in image would contain very little area, and a zoomed-out image would not preserve road shape very well, in that each pixel could easily become larger than the car.

In order to circumvent this issue, the map is not the image received from a single API request. Instead, the map is composed by several zoomed-in images from Google. This is done by dividing the original map in $4^{k_{\text{ups}}}$ parts and filling each part with a map with an adequately chosen zoom and center. k_{ups} is an integer value which can be specified to upsample the image resolution.

Figure 5 shows a close up of a part of the output image for upsampling levels $0 \leq k_{\text{ups}} \leq 4$.

There are also peculiarities involving road width on the image from Google for different zoom levels. The width is too large for low levels of zoom, and too thin for high levels of zoom. For this reason, intermediate values were chosen, such that the road has a realistic width. The values chosen for the images which compose the map correspond to this google maps view.

Figure 5: Zoom in on a roundabout in the map produced by different levels of upsampling, with $0 \leq k_{\text{ups}} \leq 4$.

Road Graph

In order to generate a path to follow only from initial and final positions, it is useful to have the road represented as a weighted graph, so that the shortest path can then be calculated.

For this end, the map was converted to a graph in a three-step process.

1. Convert the map to a binary image (1 represents desired feature, in this case, road)
2. Skeletonize the image by morphological thinning
3. Create a graph in which every 1-valued pixel is a node, and create edges between adjacent (8-connected) 1-valued pixels - the edge weights can only be 1 or $\sqrt{2}$

After this, a graph is obtained, but it has as many nodes as its corresponding path's length in pixels. This can be unwanted, and as such, one more operation was performed, which simply removes nodes that only have two connections with weight less than R, and rewires the nodes that the removed node was connected with to each other, summing the edge weights. The R parameter is used to specify minimum node density, and if left at infinity, the only nodes left will be at road intersections, destroying all information about road shape. All the graphs that go through this last step are topologically equivalent, but some information about road shape is necessary to generate a useful path.

Figure 6 illustrates the graph generation process.

Figure 6: Image, skeletonized image and graph.

Trajectory Rough Estimate

Having obtained the road graph from the image, the determination of the optimal path is much simpler, since shortest path determination in connected, planar graphs is a solved problem.

Dijkstra's algorithm is used to find the shortest path tree from the node nearest to the initial position, and the shortest path is the one from that node to the node nearest to the final position. Three examples for three different possible paths can be seen in Figure 7.

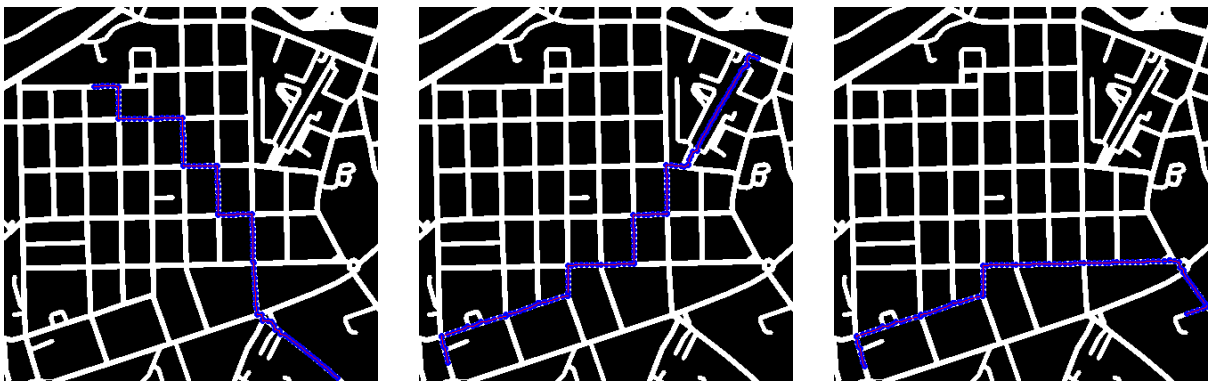

Figure 7: Shortest path for different initial and final points.

Road Limits

For physics calculations (for example, collision detection), the only relevant pieces of information are the road edges. In order to provide a representation of the road edges, a morphological operation is done on the road binary image. The image is inverted, so that 1-valued pixels represent obstacles, then all 1-valued entries whose entire 8-connected neighborhood consists only of 1-valued pixels are turned into a 0. This creates an image where only the outside borders of the road are 1-values pixels. Examples of the results are shown in Figure 8.

Figure 8: Road edges for different roads.

Performance Considerations

Running a single simulation requires many expensive-to-evaluate (in terms of time and/or computation) operations to obtain information such as a set of images from the Google Maps Static API, a graph of the road, a shortest path tree, among others.

This led to an implementation of a cache system that saves these results and reuses them in simulations where they are needed, resulting in the ability to quickly re run simulations with, for example, different controller parameters, without discarding all the previous path information.

Trajectory Generation

The function of the trajectory generator module is to take the waypoints generated by the path planner and transform them into a trajectory, that is, somehow associate time with those waypoints.

One approach to represent trajectories is to associate a time instant to each waypoint, thus defining a trajectory. However this has undesirable consequences in the context of driving a car. Namely, the car would respond to eventual delays by speeding up to compensate, trying to reach each waypoint in the planned time. This might lead to less careful driving.

To avoid this problem, the approach followed in this work was to define the trajectory as a set of waypoints with corresponding velocities. To do this, the path is divided into stretches, composed by the straight lines between each waypoint. For each stretch, a velocity is specified. The car's controller should try to keep the car's velocity at this stretch velocity and steer it along the straight line between the waypoints.

The trajectory generator has to solve the problem of what velocity to assign to each stretch. To do this, it first defines a speed limit for each stretch, based on its curvature and the speed limit of the following stretch, and then chooses a velocity for each stretch in such a way that the estimated energy consumption is bellow the desired budget and the time to reach the goal is minimized. Since the optimizer underestimates the real energy cost of controlling the car, it aims to use only a fraction of the available energy budget.

Speed limits generation

Speed limits are specified in order to prevent the optimizer from choosing velocities that are too big. The generation of the speed limits is done in two steps. First, each stretch has a speed limit assigned to it based on its curvature. Secondly, each speed limit is adjusted down if the speed limits change to quickly, that is, if they would require too much breaking acceleration to be met. Each of these conditions is explained in detail in the following paragraphs.

The limits due to the curvature of the road are based on a measure of curvature. For each stretch i, the change of direction, α_i, is defined as the smallest angle between the current stretch and the next one. Notating the length of the current path and the next one as l_i and l_{i+1} respectively, the curvature, κ_i, is obtained as

$$\kappa_i = \frac{\alpha_i}{l_i + l_{i+1}}.$$

The definition of α_i is exemplified in Figure 9.

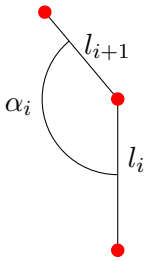

Figure 9: Values used in the definition of curvature.

The maximum velocity allowed due to curvature is determined by

$$s_{\kappa_i} = \max\left(\ v_{\max}(1 - \kappa_i G_\kappa),\ v_{\min}\right),$$

where $v_{\max}$ and $v_{\min}$ are the bounds on the curvature speed limit, set by the user, and G_κ is the curvature to velocity gain, which was set to 10.

After setting the speed limits due to curvature, the speed limits are adjusted due to a breaking deceleration constraint. Given a maximum breaking deceleration, $a_b > 0$, set by the user, the maximum speed limit at the stretch i allowed due to breaking is given by

$$s_{b_i} = \sqrt{s_{i+1}^2 + 2a_b l_{i+1}},$$

where s_{i+1} is the final speed limit of the next path.

The final speed limit for path i, s_i, is set as the lowest of the two limits, the curvature limit and the breaking limit, so

$$s_i = \min(s_{k_i},\ s_{b_i}).$$

After all the speed limits have been set, they are rounded to N_v specific values and stretches with the same speed limits are merged. This reduces the number of different stretches, which in turn lowers the computational cost of the numerical optimization detailed in the next section.

Optimizing the time of the trajectory

The velocity assigned to each stretch is set by solving an optimization problem that takes into account the maximum energy available, E_{budget}, the fraction of the energy budget that should be reserved, r_{reserve}, and minimizes the time required to reach the final waypoint.

The analysis relies on some approximations regarding the dynamics of the car. The vehicle is modeled as point mass, which means that turning does not cost energy. This approximation

allows for the analysis to consider only the lengths of each stretch, which makes the problem less complex.

The optimization variables are the velocities v_i of each stretch, joined into the vector $\boldsymbol{V}$ such that $\boldsymbol{V}(i) = v_i$. The cost that the optimizer is trying to minimize is the time taken on the path, given in terms of $\boldsymbol{V}$ and the lengths of the stretches, l_i, as

$$T(\boldsymbol{V}) = \sum_i \frac{l_i}{\boldsymbol{V}(i)}.$$

The estimated energy spent by a specific $\boldsymbol{V}$ is the sum of the difference in kinetic energies of consecutive stretches, when this difference is positive and the energy spent by the idle power. Only the positive differences are counted because breaking does not recover the energy spent. Consequently, the estimated energy consumption, $E(\boldsymbol{V})$, is given by

$$E(\boldsymbol{V}) = T(\boldsymbol{V})P_0 + \frac{M}{2}\sum_i \max(\boldsymbol{V}(i+1)^2 - \boldsymbol{V}(i)^2,\ 0).$$

The optimization problem is formulated in (7). If there were no speed limits, the problem would have a trivial closed form solution, which would be to simply have a constant speed that corresponds to the maximum velocity possible with the energy available. The existence of the speed limits makes the problem harder to solve and justifies the numerical optimization.

$$\begin{aligned} \min_{\boldsymbol{v}} \quad & T(\boldsymbol{V}) \\ \text{s.t.} \quad & E(\boldsymbol{V}) \leq E_{\text{budget}}(1 - r_{\text{reserve}}) \\ & 0 < \boldsymbol{V}(i) \leq s_i \quad \forall_i \ . \end{aligned} \tag{7}$$

The numerical optimization is performed using methods from the `optimize` module of `scipy`.

The energy budget E_{budget} is used as a parameter by the optimizer and it should be more than the minimum required to finish the trajectory. In the developed code, the user can define this parameter directly or indirectly. To define it indirectly, the user defines a velocity, v, which is used to calculate an energy budget. The indirectly defined energy budget is given by

$$E_{\text{budget}} = \left(\frac{M}{2}v^2 + \frac{P_0}{v}\sum_i l_i\right) g_{\text{est}}.$$

It corresponds to the energy the car would spend if it kept its velocity constant at v, multiplied with the gain g_{est}. This gain is necessary because the limits will force the car to brake, and so the car will need more energy. In practice, a multiplier of $g_{\text{est}} = 1.5$ has been found to be sufficient.

Since the optimizer simply approximates the energy spent, without considering the curvature of the trajectory, the complete car dynamics or the inefficiencies of the controller, the actually energy spent might be bigger. To address this issue, the energy budget E_{budget} that the optimizer receives is actually a fraction of the real energy budget. By keeping a fraction of the available energy in reserve, r_{reserve}, there can be some extra energy costs without the car failing to reach its destination.

Control

The choice of generalized coordinates in Section Car Model isn't arbitrary. Since most cars have front wheel drive, the natural control variables are the ones which are determined in the model: the front wheel's contact force and the steering wheel velocity.

The control structure used takes into account the references which are given by the trajectory generator. As seen in Section Trajectory Generation, the trajectory is a set of velocity references

associated with waypoints. Since these aren't functions of time, a problem arises concerning the determination of whether or not the waypoint reference has been fulfilled and the next waypoint reference should be given.

The velocity references given by the trajectory generator are associated with stretches of the path. Naturally, a perfect tracking of the trajectory can't be guaranteed and so the velocity reference must be associated with an area instead of just the line segment of the stretch. With such a framework, the specification of the change of reference is the specification of the line that separates the areas associated with consecutive stretches. The options considered are shown in Figure 10.

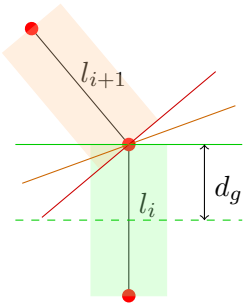

Figure 10: Waypoint crossing criteria.

These were the 3 options considered since they have geometric meaning: the separation line is perpendicular to the current stretch, perpendicular to the next stretch or it splits the angle between both stretches halfway. The line that makes the most sense is the line that is perpendicular to the current stretch, since this is the line that is nearest to the waypoint when considering the inside of the curve. This makes the car less likely to oversteer, by quickly jumping to the next reference before being close enough to the end current line segment in tight curves. The crossing criteria was chosen due to minimizing oversteering, which was verified to be an issue which caused the car to collide with walls. By minimizing oversteering, this criteria does not tend to minimize understeering, but this did not raise problems during testing.

Finally, a last parameter is added to this line: the goal crossing distance. Instead of the controller considering that the reference has been fulfiled when the car frame's origin passes this line, it can be when it crosses a line parallel to the line in Figure 10, but some distance backwards along the current stretch. This has the effect of allowing the car to start correcting its orientation, relative to the next waypoint, once its already too close to the current waypoint to meaningfully correct any error it might have. By starting to track the next waypoint further away from it, the starting orientation error is smaller, which makes the turns less sharp, improving tracking. This goal crossing distance is named d_g, as seen in Figure 10.

Steering Wheel Control

The steering wheel model is purely kinematic. Due to this, the control of the steering wheel angle is the control of a single pure integrator. Given the simplicity of the task, a simple proportional controller is used, with the control law being

$$\omega_s = K_s(\phi_{\text{ref}} - \phi). \tag{8}$$

Using the Laplace transform, with such a control law the loop gain of the controlled system is given by

$$G_s(s) = \frac{K_s}{s},$$

which, by analysis of the Root Locus, shows that for any $K_s > 0$ the closed loop poles are in the Left Complex Semi-Plane, and thus the controlled system is stable, in continuous time.

However, in any real car, the control system will almost certainly be implemented with a digital controller. For that reason, it's also relevant to study the stability of the controller, using the ideal architecture of discrete controllers: sampling the physical system's output with a perfect A/D converter and having the D/A converter perform a ZOH. In this case, the equivalent discrete transfer function of the steering system is given by

$$H_s(z) = (z-1)^{-1}\mathcal{Z}\left\{\mathcal{L}^{-1}\left\{\frac{1}{s}\frac{1}{s}\right\}|_{t=kT}\right\} = \frac{T}{z-1}. \tag{9}$$

In this case, the discrete loop gain is

$$G_s(z) = \frac{K_s T}{z-1}, \tag{10}$$

which, applying the Root Locus (which is equal in discrete time), shows that the maximum gain for which the pole remains in the unit circle is $K_s = \frac{2}{T}$. For the simulations which are ran in Section Results the sample time is 0.01 s, which means the maximum steering wheel controller gain is 200.

The steering wheel reference is determined from the trajectory reference, by attempting to aim the steering wheel towards the destination waypoint of the current stretch.

Velocity Control

In (5) it can be seen that when the steering wheel velocity is zero, the velocity dynamics are a pure integrator with a gain, corresponding to Newton's Second Law. The integrator's gain varies with the steering angle. This case, with the steering wheel velocity approximated as 0, is analyzed in detail.

The controller used was a proportional controller, with its control law being

$$F_v = K_v(v_{\text{ref}} - v_{\text{CM}}).$$

The gain from center of mass velocity to front wheel velocity is given by (3).

Given that the controlled system is, in most cases, an integrator, it is always stable in continuous time, as was seen in the previous section.

For discrete time, a similar analysis to that of the steering controller can be made, yielding the discrete loop gain of the controlled system,

$$G_v(z) = \frac{T\sqrt{\cos^2\phi + r'^2_{CM}\sin^2\phi - r'_{CM}\cos\delta\sin 2\phi}}{M(\cos^2\phi + r'^2_{CM}\sin^2\phi - r'_{CM}\cos\delta\sin 2\phi) + \frac{I_{zz}}{L^2}\sin^2\phi}\frac{K_v}{z-1}.$$

Thus, the maximum controller gain that guarantees stability is

$$K_v = \frac{2}{T}\frac{M(\cos^2\phi + r'^2_{CM}\sin^2\phi - r'_{CM}\cos\delta\sin 2\phi) + \frac{I_{zz}}{L^2}\sin^2\phi}{\sqrt{\cos^2\phi + r'^2_{CM}\sin^2\phi - r'_{CM}\cos\delta\sin 2\phi}}. \tag{11}$$

Since this depends on the steering angle, the value of the steering angle, restricted to its domain, that minimizes this function is required in order to determine a controller gain which guarantees stability regardless of the steering wheel's position. Using Mathematica™, the maximum gain allowed was determining by minimizing (11) with respect to ϕ, giving a maximum gain of 162000 for the default simulation parameters.

The dynamic component due to the steering actuation was not analyzed due to the mathematical difficulties of a rigorous treatment. However, the steering controller is assumed to be able to quickly, relative to the length of the stretch, set the desired orientation of the car. In

this condition, the car effectively moves in a straight line most of the time, after having its orientation stabilized, so the analysis shown above is valid.

Another controller, with a simpler stability analysis and a better expected performance is presented in Section Further Work, which was not simulated due to time constraints.

A remark is in order regarding this stability analysis, both for the velocity controller and the steering controller, since this analysis was done in a decoupled fashion. This means that, although in certain operating conditions, both these controllers stabilize the system's they're controlling, this was only proved under the assumption that the other system is static. Since these system's are coupled, in practice there will be moments where these conditions are not satisfied, implying that stability is not guaranteed in those cases.

A deadzone was included in the controller to prevent the controller from trying to apply very small adjustments to the velocity, in order to reduce energy consumption. The value for this deadzone is given in velocity, with the parameter v_{dead}. A v_{dead} of $0.2\,\text{m}\,\text{s}^{-1}$ means that the velocity controller does not apply any force for velocity errors with absolute value smaller than $0.2\,\text{m}\,\text{s}^{-1}$.

Parking Controller

When the car approaches the last waypoint, it has to stop and it should stop on the destination waypoint. Thus, when the reference switches to the last waypoint, the car's controller switches from tracking the waypoint's velocity to tracking the waypoint's position.

When parking, the car is only interested in the error along its x-axis. Thus, the position error is defined as

$$\varepsilon_x = x_{\text{ref}} - x_c.$$

From this position error, a kinematic controller generates a velocity error, through a proportional control law

$$v_{\text{ref}} = K_p \varepsilon_x,$$

where K_p is the kinematic parking controller's gain. This is the velocity reference which is given to the car's velocity controller. Once the car stops over the final waypoint, it is shut off.

Determination of Control Parameters

Due to the non-linearity of the system, classical control techniques for the determination of the appropriate controller gains, like the Root Locus, are not very useful. Although in certain conditions the system's dynamics can be decoupled and are linear, as seen in the previous sections, the car also operates outside those conditions. So, in order to find the best possible controller parameters for the coupled system, a search was done through the parameter space of the controller.

The controller is characterized by 4 parameters: deadzone threshold, velocity controller gain, steering controller gain and goal crossing distance.

Simulations with different values for v_{dead} were performed, and are shown in Figure 11. The graph on the left shows the energy consumption and time taken to complete an example trajectory as a function of v_{dead}. The results confirm that the deadzone successfully achieves the goal of reducing energy consumption, despite increasing the time it takes to complete the course.

The plot on the right in Figure 11 presents the fraction of the energy budget spent on different example trajectories and it also confirms that the deadzone reduces energy consumption. Note that the tests were made with an energy reserve ratio of 0.3, so the planned fraction of the energy budget to be spent is 0.7. The fraction of energy spent is close to 0.7 on most routes, as expected. On some routes it is less than that, which indicates that the energy budget given to the optimizer is high enough that it can't spend all the energy without breaking the maximum speed

limits. This explains the much smaller fraction of the energy spent on the route VerySharpTurn[1], because on this route the speed limits are very small, as the route is a single very sharp turn.

The deadzone threshold was set to $0.2\,\mathrm{m\,s^{-1}}$, because it is a small value, so it doesn't induce big velocity errors, and it reduces the energy consumption.

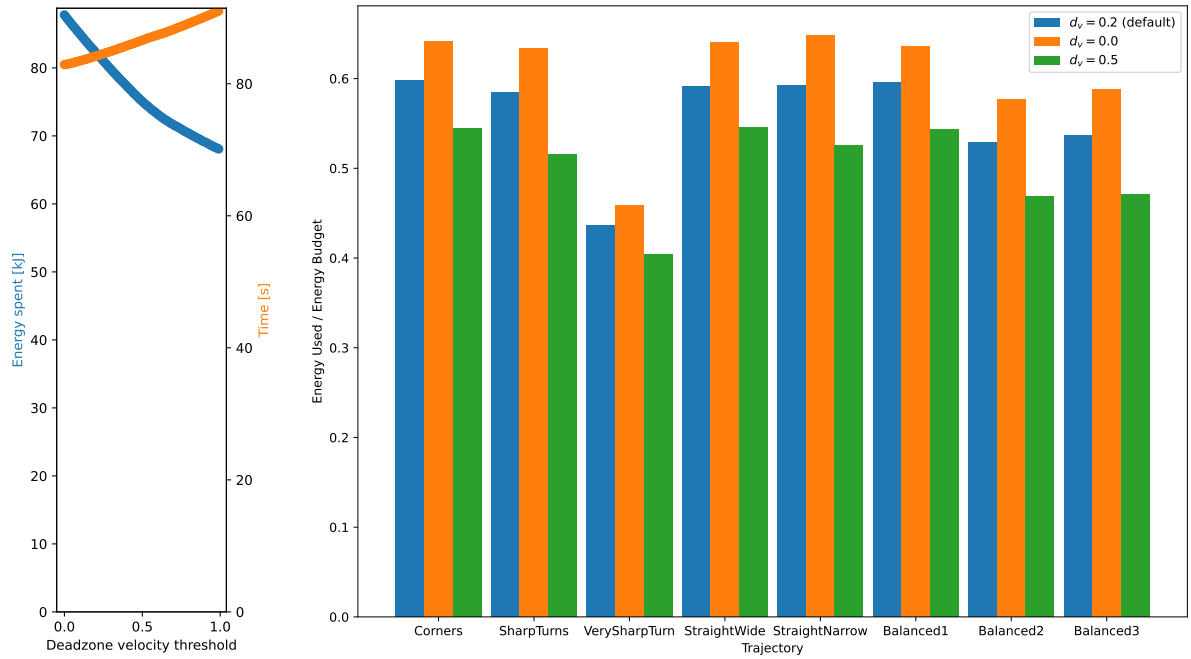

Figure 11: Total energy spent and time taken to finish course (left) and fraction of total energy spent for different courses (right), for varying values of v_{dead}, with a reserve ratio of 0.3.

The remaining parameter space was discretized, with the K_v values lying along 25 uniformly spaced points in $[100, 2000]$, K_s with 15 values along $[1, 200]$ and the goal crossing distance with 10 values along $[0, 3]$. This means 3750 controllers were tested. These ranges were chosen based on previous sparse testing, which indicated that the ideal controller is in these ranges.

In order to prevent over-fitting to a particular route, all the controllers were tested in 8 different routes, with different characteristics. Some had longer, straighter segments, while others had sharper turns. This helps guarantee that the controller chosen performs well on a wide variety of routes.

From the data gathered from each route, 6 metrics were chosen: the average tracking error, TE, the average velocity error, VE, the average absolute velocity error, AVE, the maximum power, MP, the maximum velocity actuation, MVA, and the maximum steering actuation, MSA. The tracking error is the distance between the car and the straight line connecting the waypoints of the trajectory and the velocity error is difference between the car's velocity and the velocity reference. These metrics are then averaged along the values obtained for each route to obtain the metrics for each controller.

To choose the best controller, a cost function was used that is simply a linear combination of these metrics, such that the cost of the i-th controller is given by

$$C_i = w_1 \mathrm{TE}_i + w_2 \mathrm{VE}_i + w_3 \mathrm{AVE}_i + w_4 \mathrm{MP}_i + w_5 \mathrm{MVA}_i + w_6 \mathrm{MSA}_i.$$

The choice of the weights of this function determines on what is most important for the what the best controller should be. The weights used are $w = [1\ 0.5\ 0.5\ 1\ 1\ 1]^T$, since they made

[1] A video showing this route is available at https://web.tecnico.ulisboa.pt/ist193152/default_controller_verysharpturn.mp4

sense from an intuitive viewpoint, prioritizing keeping to the trajectory and minimizing energy and actuation over following the reference velocities perfectly.

Additionally, besides the cost function, all controllers which had collisions were excluded from the potential candidates. Thus, minimizing this cost functions among the controllers without collisions, led to the controller

$$K_v = 733.33 \qquad K_s = 15.21 \qquad d_g = 2.54\,\text{m}. \tag{12}$$

Since this controller satisfied all the requisites, keeping to the energy budget, having no collisions with the sides of the road and being sufficiently fast, the weights of the cost function were deemed acceptable and this controller is the one used throughout the rest of this work. The results of this controller, along with comparisons to other controllers, can be seen in Section Results, in particular in Figure 13 and Figure 14.

Simulator

A simulator for this system was implemented in the Python programming language. For the duration of the simulation, t_{sim}, the controller samples the system every t_{step} seconds, and the car is simulated for the same duration.

The car's energy spending is monitored. If the energy spent exceeds the energy budget, the car brakes until it comes to a stop.

The car follows the given trajectory, and simulation stops after either the car's velocity is approximately zero for t_{obs} seconds, or if it takes longer than t_{sim}.

The simulator's general architecture is shown in Figure 12.

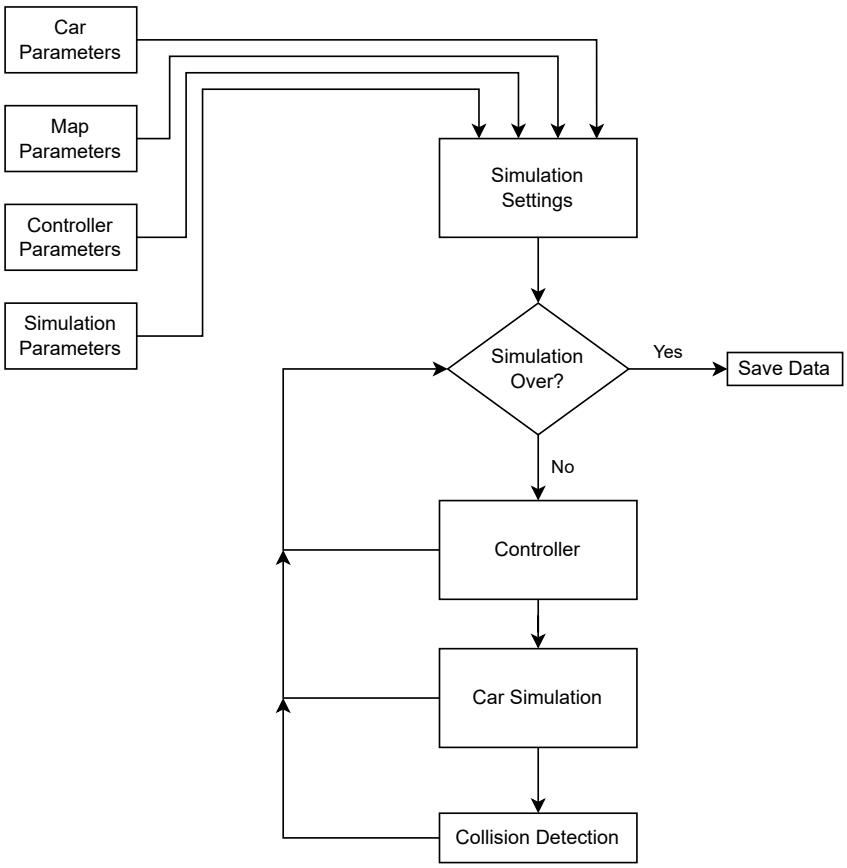

Figure 12: Diagram of the simulator architecture.

Parameters

The default parameters used for simulation purposes are presented.

Table 1: Default simulation parameters.

Maximum simulation time t_{sim}	Controller sample time t_{step}
100 s	10 ms
Simulation time after imobile car t_{obs}	Energy budget E_{budget}
5 s	Define through speed: $10\,\text{m}\,\text{s}^{-1}$

Table 2: Default car parameters.

L	L_{r}	L_{f}	d
2.2 m	0.566 m	0.566 m	0.64 m
r	Length	Width	Mass
0.256 m	3.332 m	1.508 m	810 kg
Center of mass r_{CM}	Center of mass δ	Moment of Inertia I_{zz}	Idle power P_0
1.1 m	90.0°	$2080\,\text{kg}\,\text{m}^2$	500 W

Table 3: Default controller parameters.

Steering controller gain K_s	Engine velocity controller gain K_v
15.21	0.566 m
Goal crossing distance d_g	Velocity tracking deadzone v_{dead}
2.54 m	$0.2\,\text{m}\,\text{s}^{-1}$

Table 4: Default trajectory generator parameters.

Curvature max speed limit	Energy estimation gain g_{est}	Latitude
$30\,\text{km}\,\text{h}^{-1}$	1.5	38.736 725 6°
Curvature min speed limit	Energy reserve ratio r_{reserve}	Longitude
$10\,\text{km}\,\text{h}^{-1}$	0.3	−9.138 887 1°
Maximum deceleration a_b	Waypoint smoothening window	Base map zoom
0.1 g	5	16
Number of speed limits N_v	Graph regularization R	Upsampling k_{ups}
10	5	3

Collision Detection

The simulation environment contains only the road and the car. Collisions are detected between the car and road edges using the Separating Axis Theorem.

Both the car and the road edges are decomposed in a collection of convex polygons. The car is decomposed in four rectangles: the rear wheels, the body and the front wheel (for this application's purposes, the front wheel is redundant for collision detection, as it is always inside the body). The road edges are decomposed in as many squares as there are pixels representing the road edges. The side of these squares is the scale s of the map (in meters per pixel). According to Google, s can be calculated as

$$s = \cos(lat) \times 156543/(2^{zoom+k_{\text{ups}}}) = \cos(lat) \times P_E/(2^{8+zoom+k_{\text{ups}}}),$$

where *lat* is the latitude, P_E is the earth circumference, *zoom* is the base map zoom and k_{ups} is the upsampling constant.

The normal vectors to the face of each rectangle are used as separating axes. The objects are colliding if and only if the projections along any of the separating axes overlap.

Results

The tracking error for the velocity and distance, along with the actuation for both the car velocity and steering wheel angle, are shown in Figure 13, for different values of K_v, and in Figure 14, for different values of K_s.

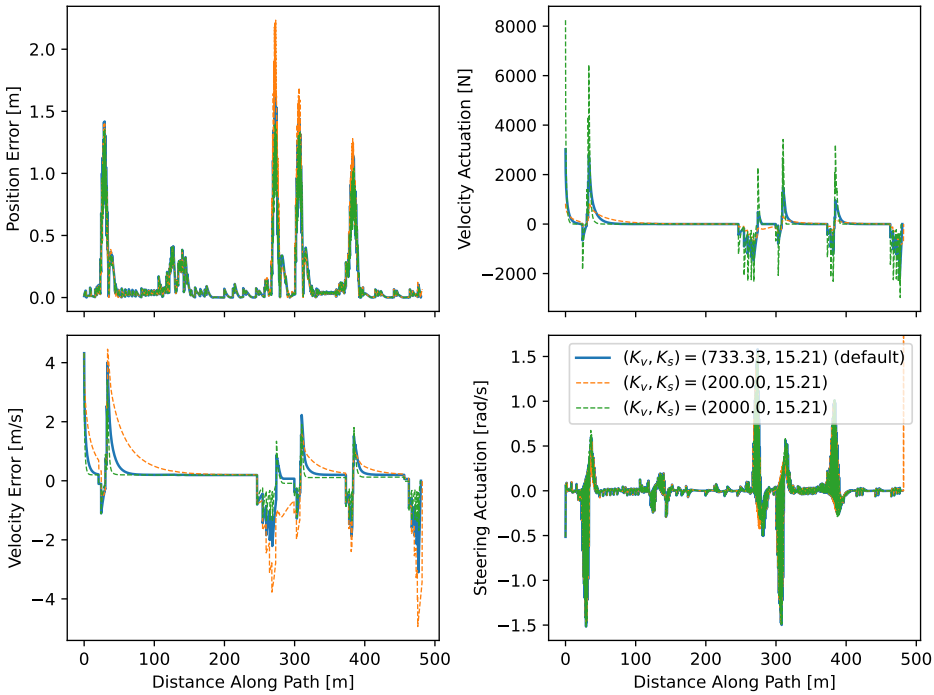

Figure 13: Position and velocity tracking error and velocity and steering actuation for different values of K_v.

It is possible to observe that, for a lower gain K_v, the position error does not change significantly. However, the velocity error is increased to a great extent. In the case of a much higher gain, the position error also remains unchanged for the most part, and although the velocity error does diminish, this comes at the substantial cost of large actuation values. No collisions occurred for any of these controllers. Energy spent increased with the increase of K_v: $E(K_v = 200.00) \approx 73.9\,\text{kJ}$ and $E(K_v = 733.33) \approx 82.3\,\text{kJ}$, $E(K_v = 2000.0) \approx 89.0\,\text{kJ}$.

For a small K_s, a large delay on the steering controller is observed. In the beginning of the simulation, the path is a straight line. As soon as it has to turn, it fails to do so due to the aforementioned delay, creating the visible oscillations on both the position and the velocity tracking errors.[2]

A large K_s presents the same issue as in the case of a large K_v - large actuation values. In this case, however, there is no significant gain in terms of tracking errors.

In the simulation performed with $K_s = 1.00$, collisions occurred. For $K_s = 15.21$ and $K_s = 150.00$, there were no collisions. The energy spent for the cases without collisions are

[2]A video showing the results with this controller showing the oscillations, in a different route, is available at http://web.tecnico.ulisboa.pt/ist193152/low_ks_controller_verysharpturn.mp4.

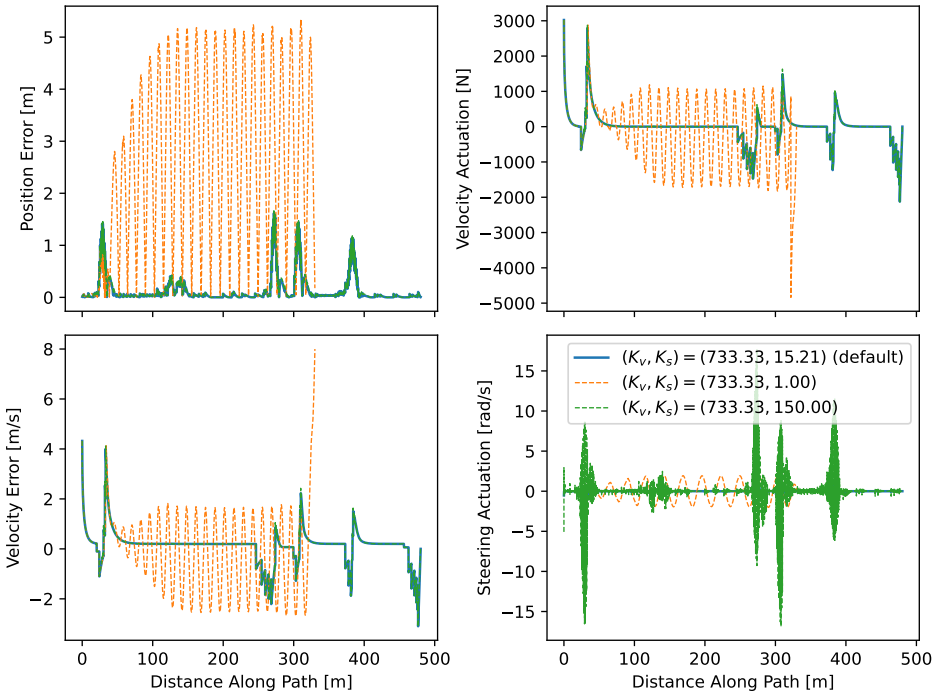

Figure 14: Position and velocity tracking error and velocity and steering actuation for different values of K_s.

roughly equal: $E(K_s = 15.21) \approx 82.3\,\text{kJ}$ and $E(K_s = 150.00) \approx 82.3\,\text{kJ}$. This is in part due to the massless dynamics model of the front wheel.

Figure 15 shows the behaviour of the system for a gain $K_v = 180\,000$, which the theoretical analysis correctly predicts to be unstable.

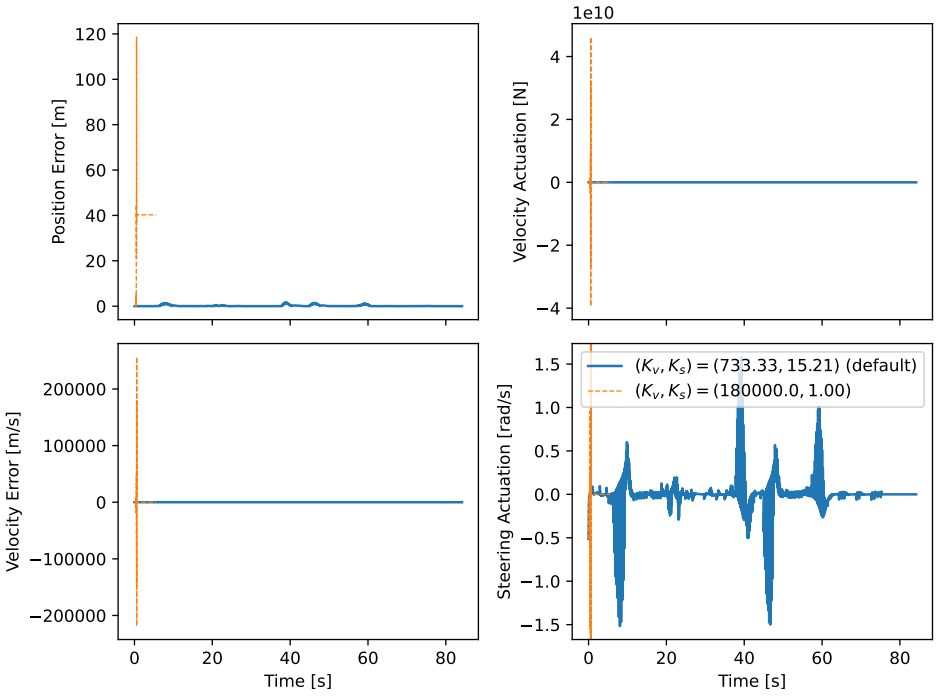

Figure 15: Example of an unstable controller.

Conclusions

Simplified kinematic and dynamic models of a car were derived. A simulation environment was implemented with automatic road data retrieval, path planning and velocity optimization to minimize travel time constrained by an energy budget. Proportional controllers of velocity and orientation were designed and tuned based on simulation data.

The resulting car is able to autonomously plan a trip between two points and follow it, in a simplified simulation environment, without moving obstacles. Example videos showcasing the visual output of the simulator are available at the following links: https://web.tecnico.ulisboa.pt/ist193152/default_controller_corners.mp4, https://web.tecnico.ulisboa.pt/ist193152/default_controller_balanced1.mp4

Further Work

Some further improvements to the work developed are now discussed.

Velocity Controller

A different controller, which was not implemented in this work due to time constraints, is one which linearizes the velocity dynamics through state feedback. The velocity dynamics from (5) can be compactely written as

$$\dot{v}_f = f(x) + g(x, \omega_s) F_v,$$

where x is the car's state.

Since ω_s is an input and the steering controller can determine which value to apply independently of the car velocity, at the time of determination of the force to be applied by the car's velocity controller, this value can already have been determined. This means that the velocity controller can simply be

$$F_v = \frac{-f(x) + u}{g(x, \omega_s)},$$

where F_u is some force to be applied. By doing this, the velocity dynamics become

$$\dot{v}_f = u.$$

Then, a simple linear controller can be used on this system, that is a pure integrator, yielding simple analysis and a very easy to stabilize system.

Sensor Simulation

In this work, in order to limit the scope, the state measurements were simply the state, implicitly assuming that this could be measured. In a real car, the state components would be measured by sensors, which in some cases have a lot of noise, for example, GPS or odometers. To deal with this, a state estimator would be introduced, probably a Kalman Filter.

Local Sensing and Real-time Trajectory Planning

As a consequence of assuming that the environment was static, local measurements were disregarded, with the work developed dealing with the global state. However, to further improve this system and bring it closer to applicability in real life, local measurement sensing would be added. An example is the measurement of distances to surroundings, to avoid collision with other vehicles or unexpected obstacles.

When the assumption of a static environment breaks, trajectory planning also has to be turned into a real-time endeavor, in order to accommodate the possibility of changes to the possible paths.

Appendix A - Software Documentation

The software developed can be roughly separated into 3 parts:

- The simulator, which is a simple simulator with a fixed discrete time step. The modeled modules are defined as continuous time dynamical systems. Thus, they expose two important functions: the derivative function, which relates the derivative of the state with the current state and inputs, and the output function, which relates the output with the current state (For simplicity of the simulation the output was not allowed to depend on the current input. This did not cause issues since these systems naturally already had this property). Through the choice of an adequate time step for the simulation this method proved sufficient and did not lead to numerical stability problems. A diagram describing the simulator's can be seen in Section Simulator.
- The autonomous car system, which is composed by various components: the car model, the path planner, the trajectory generator and the controller.
- The testing suite which both allows validation of the rest of the software and the determination of parameters for the controller, by testing various combinations of the these parameters and optimizing for a certain cost function.

Running the Simulator

Running the simulator allows the user to see a real-time simulation of the car in an environment which may specified by the user, with a complete graphical interface which shows both the movement of the car as the remaining interest variables: current velocity, energy and number of collisions.

To run the simulator, the user must merely run the command:
`python3 simulator.py`
The simulator also produces a `simulation.mp4` video file so the simulation can be seen again afterwards.

Choosing different settings for the simulator

The simulator settings are all gathered in a `SimSettings` class, details of which can be altered in its instantiation in `simulator.py`.The parameter list is quite extensive and can be fully seen in `sim_settings.py`, accounting for each configurable element of the simulation.

Running the Testing Suite

The testing suite is a module with 3 important submodules:

- The controller tests, which produce error plots for a variety of controllers, including the default one.
- The deadzone tests, which produce a series of plots of energy spent vs time, to evaluate the impact of the controller velocity deadzone in diminishing the energy spent while simultaneously tracking the impact in the error. No simulations with collisions are admissible and they are not counted.
- The controller parameters mass tests, which runs a very long simulation (≈8 hours) and takes up a lot of space due to the cache (≈30Gb). This runs simulations that test all combinations of the important controller parameters (the goal crossing distance, the velocity control gain and the steering control gain), each within a discrete number of options. Each

of these controllers is tested on a set of different trajectories, with different path characteristics (sharper turns, larger straight stretches, etc.), then measuring a set of relevant metrics, like the average tracking error, average velocity error, etc. These are then used to determine a cost for each controller. The controller that minimizes this cost function is considered the optimal controller. The default controller for the simulator is the one determined through this mean.

To run each of these tests, the user must run the following command:
`python3 -m testing.[controller_tests/deadzone_tests/mass_tester]`

Requirements

In order to run this software, some external python libraries are required. To install them simply run:

```
pip3 install -r requirements.txt
```