Planning and Motion Control in Autonomous Heavy-Duty Vehicles

RUI OLIVEIRA

KTH Electrical Engineering

Master's Degree Project
Stockholm, Sweden September 3, 2014

XR-EE-RT 2014:021

To my parents

Acknowledgments

Thanks to my supervisor Jonas Mårtensson for guiding me in the correct direction during the long journey that was this thesis. To my co-supervisor Pedro Lima a big thank you for providing help whenever it was needed.

Most of all I thank my parents, Jorge and Maria, for everything they did for me. Thank you for everything, for always placing your kids in front of your own needs, for the sacrifices endured to make my life easier and meaningful. Your love and caring resulted in the good qualities that I have today, and certainly your tough love also eliminated some of the less good characteristics that I could have had.

To João and Maria, a big loving hug. Your friendship is one of the most valuable things I am proud to have. Thank you for all the good moments that we shared, it's still hard for me to believe the luck I had to meet you both. In you I have found an extra motivation to excel in my studies and also the importance of slowing down and enjoy life.

Thank you Diogo for these last two years of studies. Thanks to you I jumped from a good student to a great student. You did more than just helping me in this jump, you pushed me forward and even carried me whole in the tough times. Thank you for all the patience that I often required.

To Carolina and José a hug and a kiss, respectively, your young spirit and fresh personalities managed to bring so many laughs to otherwise boring moments. Thanks for intoxicating me with that teenager joy that I was starting to lose.

To every friend in Portugal that helped me in way or another, your smiles, kindness and friendship deserve much more than these mere words.

Finally to all of the amazing friends I had the chance to meet during my stay in Sweden, a big *skål*, you made this year the best of my life.

Abstract

Autonomous driving has been an important topic of research in recent years. In this thesis we study its application to Heavy Duty Vehicles, more specifically, vehicles consisting of a truck and a trailer. An overview study is done on three fundamental steps of an autonomous driving system, planning, trajectory tracking and obstacle avoidance.

In the planning part, we use RRT, and two other variants of the algorithm to find trajectories in an unstructured environment, *e.g.*, a mining site. A novel path optimization post-processing technique well suited for use with RRT solutions was also developed.

For the trajectory tracking task several well-known controllers were tested, and their performance compared. An extension is proposed to one of the controllers in order to take into account the trailer. The performance evaluation was done on scaled truck systems in the Smart Mobility Lab at KTH.

The obstacle avoidance is done with the aid of a simple, yet functional Model Predictive Controller. For this purpose, we developed different formulations of the optimization problem, corresponding to distinct optimization goals and vehicle models, in order to assess both the quality of the MPC, and of the assumed truck model.

The outcome of this thesis is a fully autonomous system, able to plan and move in constrained environments, while avoiding unpredicted obstacles. It was implemented using a 1:32 scale remote controlled truck, commanded by a desktop computer.

Contents

Chapter 1

Introduction

In this Master Thesis we will study the feasibility of implementing an autonomous driving system applied to Heavy-Duty Vehicles (HDVs) moving in unstructured environments.

This topic is related to the iQMatic project which will now be explained.

1.1 iQMatic Project

The iQMatic project is an attempt to reach autonomous driving in HDVs. This project is a collaboration between Scania, Saab, Autoliv, and the universities KTH Royal Institute of Technology and Linköping University. A first application of these autonomous system would consist of industrial application such as rock quarries, mine exploration, construction sites and factory environments. In this type of environment one is usually constrained to off road driving, we will call these regions without road networks as unstructured environments.

The main goal consist in achieving transportation of goods from one place to another, the use of HDVs instead of lighter and smaller vehicles results in an increase of the transportation capacity and fuel efficiency. The transportation task is usually repeated several times, thus becoming a repetitive and dull task for human drivers. Since many accidents are related to driver fatigue and distraction [1], the automation of said tasks would result in an high safety and reduction of accidents. A large slice of driving costs is related to driver salaries and fuel consumption, automating this process would also result in monetary benefits [2].

The whole system can be roughly divided into the following sections:

Path planning

In an initial stage it is necessary to find a way to a desired destination from a current position. If we are in a road network, this would consist of finding a sequence of roads and streets that, if followed by the vehicle, will make it arrive at the destination. If the environment is an unstructured one, without roads, then the planning will consist of a set of waypoints

that guide the vehicle to its destination. Finding the sequence of roads or waypoints is not a trivial task, and it can be made even harder if there are secondary objectives, like fuel or time efficiency.

Trajectory tracking
The trajectory tracking is concerned with the control of the vehicle, such that it performs the necessary actions to follow the waypoints and arrive at the goal. Due to the required safety of the vehicle and of the elements surrounding it, it is of extreme importance that the trajectory tracking is well executed.

State estimation
In order to have a good driving performance it is necessary to measure physical parameters of the vehicle, such as its current position, velocities and even inertial parameters which might change dramatically for different cargos, and that can be hard to measure. The state estimation is a fundamental piece for the control algorithms employed in trajectory following, and as such, it should be taken into account with the same importance as the trajectory following.

Sensing
Sensing and Estimation can seem similar to the reader. Here we will assume that Sensing is concerned with more external aspects of the environment, while Estimation focuses on internal properties of the system. The topic of sensing is very wide, and it changes according to the application at hand. In the case of driving in a road network, the sensing would be concerned with knowing the whereabouts of other drivers, or even pedestrians crossing the street. In a more unstructured environment, sensing can refer to noticing obstacles in the way, *e.g.*, large debris in the trajectory.

Obstacle avoidance
Obstacle Avoidance will work mainly with the Sensing block, asking it if there are any imminent threats for the vehicle, such as obstacles or other cars. If an obstacle is detected and there is a prediction of a collision in a near future, the obstacle avoidance will assume control of the vehicle, while trying to take it out of any dangerous situation.

1.2 Smart Mobility Lab

The Smart Mobility Lab is a KTH facility belonging to the Automatic Control Department. This laboratory is intended to be used for a wide range of experiments. One of its main features consist in a high-precision motion capture system, which is able to provide faithful pose estimates of rigid bodies moving inside the laboratory.

Using the motion capture system, control, estimation and sensing algorithms can be tested for several types of vehicles, such as scaled trucks, [3],[4] and [5], unicycle like robots [6] and quadcopters [7] and [8]. In our case we will make use

of the motion capture system to test the developed autonomous system when applied to 1:32 scale trucks.

1.3 Project Outline

The developed work is composed of three different parts:

- Planning
- Trajectory Tracking Controllers
- Obstacle Avoidance

In the planning part we will solve the problem associated with finding a path that achieves the desired destination. To this end we will implement and compare three different algorithms from the literature, these algorithms are all sampling-based, sacrificing quality of solution for an increase in permormance. The algorithms are adapted to the particular situation of an HDV, and multiple parameters and heuristics are studied. A novel optimization algorithm is also developed, which seeks to improve the quality of found paths through the use of machine learning techniques.

The found path must now be performed and to do so we make use of controllers. In this section we study on trajectory tracking algorithms, wich focus on following the way points with a certain temporal rule. There is in the literature a large number of solutions to this problem, specifically when applied to car-like vehicles. The truck and trailer configuration has not yet been well studied and as such, this project will try to make use of the already available knowledge for simple vehicles and apply it, when possible, to the truck and trailer case. The controllers will be tested making use of the scale trucks available in the Smart Mobility Lab.

The finishing topic of this thesis is concerned with obstacle avoidance, which seeks to deal with the problem of unexpected obstacles or events. Ideally the vehicle would be able to deal with an abnormal situation, such as a car crash in the middle of the road, or a parked car in an open space. To deal with this situation, the vehicle must avoid the obstacle, and proceed again with the normal trajectory tracking. For this purpose we opted to implement a model predictive controller, that is able to perform the trajectory tracking task while avoiding previously unknown obstacles, either static or moving.

Chapter 2

Planning

In this chapter we address the planning part, which represents one of the first steps in most autonomous systems. Section 2.1 will explain the problem to be dealt with, while enumerating the details and particular difficulties that arise for the specific situation we are in. In section 2.2 the Rapidly-exploring Random Tree (RRT) algorithm is explained, and its performance evaluated. Section 2.3 will present a new formulation of the RRT algorithm, which attempts to decrease the underlying stochastic nature of the original algorithm. Section 2.4 presents yet another version of the RRT algorithm, the RRT*, unlike the previous algorithms, this version, guarantees that the found path will converge to the optimal, given enough time. A novel local path optimization algorithm was developed and is detailed in section 2.5. This optimization procedure, based on genetic algorithms, was designed so that it is suitable to be applied after an RRT planning phase.

2.1 Problem Statement

2.1.1 The need for planning

The main objective of an autonomous driving system is to achieve transportation from one point to another. One of the initial steps is then to find a way to arrive at the destination, starting from the current position.

Two different classes of the problem can be identified. The first happens when we can arrive at the destination through a road network. In this case, it is usually assumed that the road links are known, and finding an optimal (or close to optimal) path becomes a well known problem. A common example of this are the recent GPS solutions found in cars, which are able to indicate to the driver a set of instructions that will take him to the desired destination. Another example is the Google Maps application, which provides its users with a very sophisticated route planner, detailing several possible ways for a driver, or even a pedestrian, to arrive at a destination.

The second class, and the one related to the situation we are tackling occurs

when the environment does not include roads or any other kind of structured pavement. This situation is fairly common in construction sites and natural resources exploring venues. In this case one assumes that we have access to a map containing the spatial information relating to the travelling zones (where one can drive) and the obstacle zones (where driving is not allowed). Alternatively to have a global map, one might only be able to measure the environments free zone in its close surroundings.

The main goal of the planning part is to arrive at a particular place. One can decide that this place corresponds to a goal region, or to a particular pose. Arriving at a goal region is by far a much more relaxed objective than a particular pose.

The secondary goals of a planning algorithm are usually related to a desirable efficiency of the found solution. This efficiency can be of several different types, and it is usually adjusted in order to meet the needs of the truck owner. A very common efficiency measure is the distance travelled to arrive at the goal, by making it as small as possible one often achieves time and fuel savings (although not always). Due to the constant and repeated use of the trucks, this efficiency becomes a very desirable characteristic, since one would expect huge monetary savings in the long run.

Planning algorithms must also obey to several constraints, depending on the type of the problem to be solved. These constraints will be addressed next.

For a comprehensive tutorial on the planning problem, the reader is referred to [9], a whole book is also dedicated to this problem in [10].

2.1.2 Assumptions/Problem Statement

Having defined some of the possible scenarios with which a planning algorithm might have to deal we will now specify the problem we are going to address. By doing so, we can adapt/choose the algorithm to the particular situation we are facing.

We will assume that we have *a priori* knowledge of the environment and that the driving region is limited by some known boundaries, the interior of these boundaries defines the workspace. Additionally we know the position and region of all obstacles. By knowing the obstacle positions we immediately know the free zones, which are simply the remaining of the workspace. Figure 2.1 shows the example of a possible environment configuration.

The problem to solve can be defined as follows. Given an initial truck configuration, corresponding to the initial pose of the truck in the free zone, arrive at a goal region, also in the free zone, by travelling inside the workspace while avoiding the obstacles. The obstacles are assumed to be static, and will not change its position during the execution of the planning and the travelling. The obstacles are assumed to be represented by polygons (the need for this restriction is explained in 2.2.2).

We decided to define the objective destination as a goal region, since it greatly simplifies the problem. If the objective would be a truck pose, the algorithm would have to put an extra effort into finding the solution. This

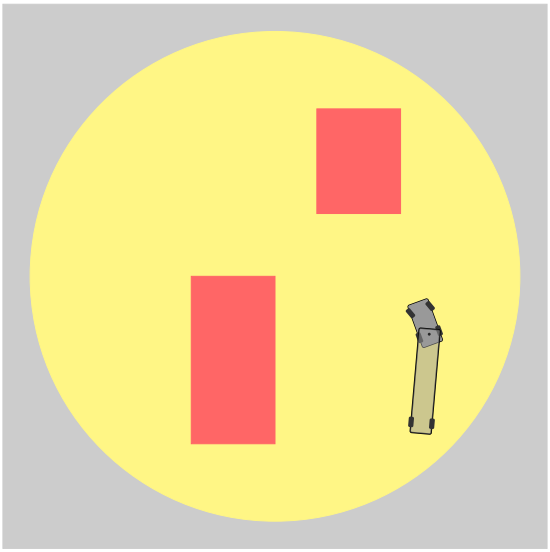


Figure 2.1: Example of a possible environment. The workspace consists of the free zone, on yellow, and of obstacles on red. The grey area is outside of the workspace boundaries, and driving is not allowed on it

extra effort is usually related to the need to perform complex manoeuvres such as reverse driving and/or parallel parking-like movements.

2.1.3 Main challenges

Different planning problems often have different difficulties. By identifying and understanding the particular problems arising in the situation we are dealing with, one can better choose and use a planning algorithm suited for the problem at hand.

Nonholonomic constraints

One of the major difficulties in planning for the cars (and a great number of other systems) is due to the nonholonomic constraints arising from the kinematic vehicle model.

The kinematic model expresses the truck evolution, as a function of the inputs applied and the current state. For the truck and trailer system, it is as follows [11]:

$$
\begin{aligned}
\dot{x} &= v_1 \cos\theta_1 \\
\dot{y} &= v_1 \sin\theta_1 \\
\dot{\theta}_1 &= \frac{v_0}{L} \sin\left(\theta_0 - \theta_1\right) \\
\dot{\theta}_0 &= \frac{v_0}{R} \tan\phi
\end{aligned} \tag{2.1}
$$

Where the state variables are defined as in figure 2.2. x and y correspond to the trailer axle position, while θ_1 and θ_0 correspond to the trailer angle and truck angle respectively.

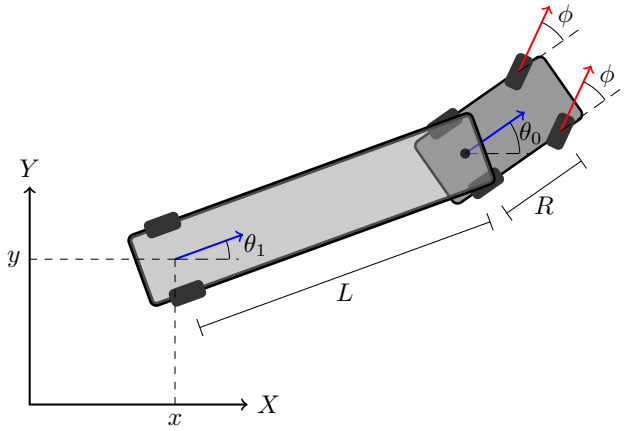


Figure 2.2: State variables of the truck and trailer model

v_0 corresponds to the linear velocity of the tractor, and it is along the θ_1 direction, while v_1 is the linear velocity of the trailer, along the θ_0 direction. The inputs commands are v_0 and ϕ and correspond to the linear velocity and steering angle respectively, the additional velocity of the trailer is computed as $v_1 = \cos\left(\theta_0 - \theta_1\right) v_0$. The final objective of a planning algorithm is to find a sequence of input commands $(v_0(t), \phi(t))$ that makes the truck and trailer arrive at the goal region.

The nonholonomic constraints arise from the kinematic model and have the effect of not allowing the system states to move freely in arbitrary directions. A simple example can be given if we imagine the car system (which has a kinematic model similar to one expressed in (2.1)). The car system is not able to move sideways, it is only capable of moving in straight lines or curved segments, in the forward or backward direction.

Figure 2.3 shows the initial state of a car and the objective pose (in a lighter shade). Due to the nonholonomic constraints it is impossible for the car to simply move sideways, as the dashed path in red shows. One possible solution is the one shown in green, which obeys to the kinematic model, that is, it is composed of straight lines and curved segments in the forward direction.

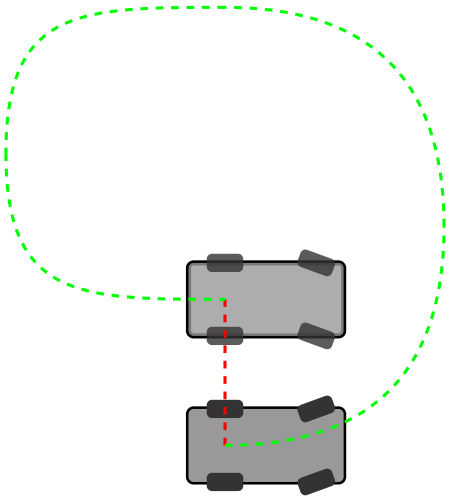


Figure 2.3: Example of car's holonomic constraint, it is impossible for the car to move sideways (red line), a possible way to achieve the same translation consists in the green path

After showing these examples the reader should be convinced that the non-holonomic constraints constitute a serious difficulty in the planning process, and must be carefully dealt with.

Obstacle Avoidance

In order to travel safely the vehicle must avoid possible obstacles that might lie in the environment. In the case when there are no obstacles in the environment (and the environment has a large enough workspace) the problem becomes a well known one. The solution can be found through Dubins' curves [12], which correspond to a well established technique of finding paths with the shortest distance for car-like vehicles.

Due to the existence of obstacles in the environment, said technique cannot be used, and as such alternative methods must be used. Figure 2.4 illustrate the obstacle avoidance problem.

The objective is to move from the right position to the left one. However the presence of the obstacle must be taken into account. The red path in the middle is not possible, since it crosses the obstacle. The bottom path, although not crossing the obstacle, would result in a collision, since the vehicle width is bigger than the margin with which the obstacle is avoided. The green path represents a possible solution, since it avoids the obstacle with a large enough safety margin, that is, the vehicle is never in collision with the obstacle, while it is travelling the path. It then becomes clear that in order to successfully avoid obstacles, one must also take into account the vehicle dimensions.

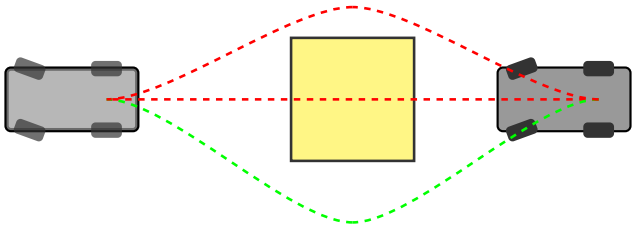


Figure 2.4: Going from right to left in the presence of an obstacle. The red paths represent unfeasible solutions, while the green path shows a possible solution

Jackknifing

Jackknifing is a situation that appears due to the existence of a trailer attached to the truck. It happens when the difference between the trailer orientation and the truck orientation becomes too large, as seen in figure 2.5.

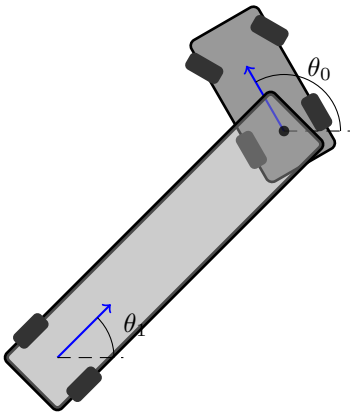


Figure 2.5: Jackknifing happens when the difference between θ_0 and θ_1 is too large

This problem usually happens when in a reverse manoeuvre. In that situation, the vehicle system becomes unstable, and the truck has a tendency to jackknife.

However this problem can also happen if the truck is required to curve very hard for a long period of time.

Steering Limitations

One last limitation worth mentioning is related to the maximum steering angle of the truck. This happens because the angle of the steering wheels is limited to a certain value, thus resulting in the truck only being able to turn with a curvature up to a certain minimum radius, which is given by

$$R_{min} = \frac{R}{\tan \phi_{max}} \tag{2.2}$$

Having a smaller steering angle will result in a lower manoeuvrability, since the maximum curvature will be smaller.

2.2 RRT

In order to address all of the problems mentioned earlier we chose to make use of the Rapidly exploring Random Trees algorithm[13]. This algorithm has been widely used in the last years, and is well documented in the literature. One of its main advantages comes from the fact that it incorporates the nonholonomic contraints in a simple way, and that it is supposedly able to solve difficult problem instances quickly.

On the other hand its biggest disadvantage is related to the stochastic nature and highly sub optimal solutions that result from the algorithm.

Several other algorithms could have been used to solve the planning problem, depending on the desired requirements of the user. For a very fast planning algorithm, able to plan in constantly moving environments, one could have used [14]. In [15] vehicle dynamics more complex than the ones expressed by the kinematic model are taken into account, this becomes important in high-speed driving. Another possible planner is presented in [16], and it is concerned about minimizing the uncertainty often present in planning.

2.2.1 The algorithm

The RRT original implementation can be found in [13]. Its explanation can be given in a simple way through algorithm 1.

$\mathcal{T} \leftarrow$ `CreateTree(`x_{init}`);`
while *not at goal* **do**
  $x_{rand} \leftarrow$ `RandomState();`
  $x_{near} \leftarrow$ `NearestNeighbour(`x_{rand}`,`$\mathcal{T}$`);`
  $(u, x_{new}) \leftarrow$ `Steer(`x_{rand}`,`x_{near}`);`
  $\mathcal{T} \leftarrow$ `AddNode(`x_{near}`,`x_{new}`,`u`);`
end

Algorithm 1: RRT algorithm

A brief explanation of the steps is now given.

This first line is responsible for initializing the tree structure. The tree is simply created with its root node corresponding to the initial configuration of the vehicle.

The algorithm will then run the `while` loop, with the following procedures:
$x_{rand} \leftarrow$ `RandomState`

This line simply makes the state variable x_{rand} equal to a randomly drawn state in the workspace. This state is not in obstacle collision or in a jackknife situation. With a probability of p_{goal} the output x_{rand} will not be a random one, but instead a configuration in the goal region or equal to the objective pose.

$x_{near} \leftarrow$ `NearestNeighbour`$(x_{rand},\mathcal{T})$

This function receives the previously computed x_{rand} and will find the closest node in the tree $\mathcal{T}$. This closest node correspond to the node with state configuration that is closest to x_{rand}. The measure of nearness can be decided by the developer, a typical choice is the euclidean distance.

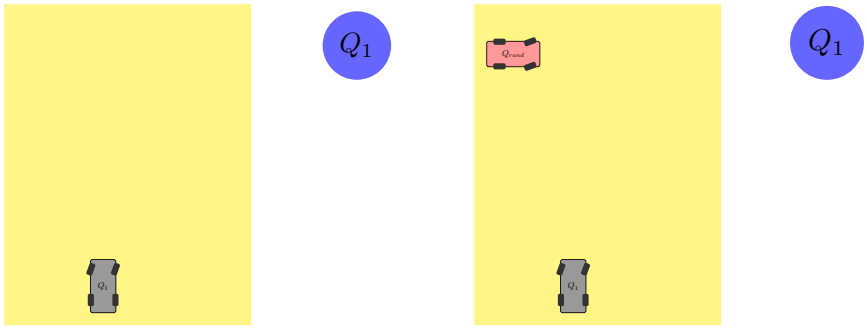
$(u, x_{new}) \leftarrow$ `Steer`(x_{rand},x_{near})

`Steer` attempts to steer the system from the nearest node x_{near}, in the direction of x_{rand}. The objective is to steer in the direction of x_{rand}. We limit the amount of distance that `Steer` can move, so usually it will not achieve x_{rand}. If the performed movement causes a collision with an object, the function will simply cause the `while` loop to jump to the next iteration. Otherwise it will return the new achieved state x_{new} and the necessary input u to arrive at it from x_{near}.

$\mathcal{T} \leftarrow$ `AddNode`(x_{near},x_{new},u)

If the loop arrived at this function, then the `Steer` function successfully found a new state x_{new}, which will then be added to the tree $\mathcal{T}$ as a child of x_{near}. The additional information of the input command u is also associated to the link connecting the nodes.

Figures 2.6 to 2.10, show an example of the execution of the algorithm.

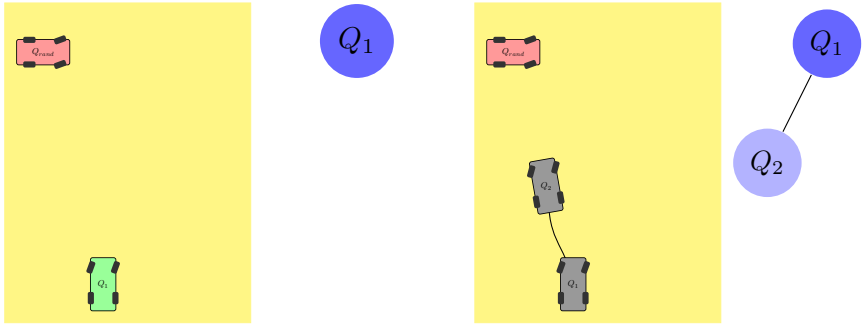


(a) Initial configuration of the vehicle, and its node Q1 (root) in the tree

(b) Sampling a random vehicle configuration in the free space

Figure 2.6

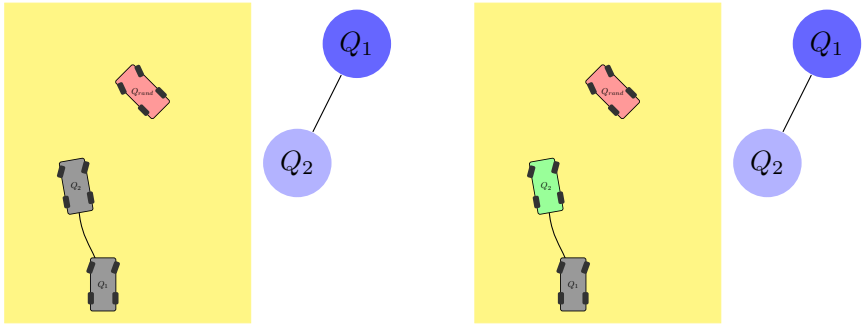
The illustrated situation did not show any possible collision, however, one can simply imagine that if an obstacle somehow is in the way of the local movement found by `Steer`, that current iteration will just stop, and no new node will be added to the tree.



(a) Finding the state in the tree nearest to the random vehicle configuration

(b) Steering the nearest node in the direction of the random state

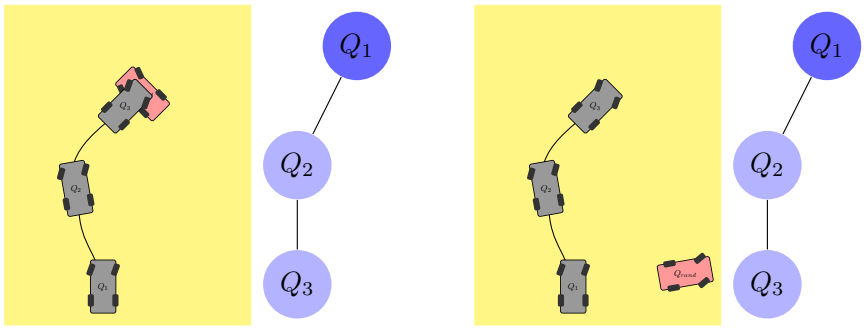
Figure 2.7



(a) Sampling a random vehicle configuration in the free space

(b) Finding the state in the tree nearest to the random vehicle configuration

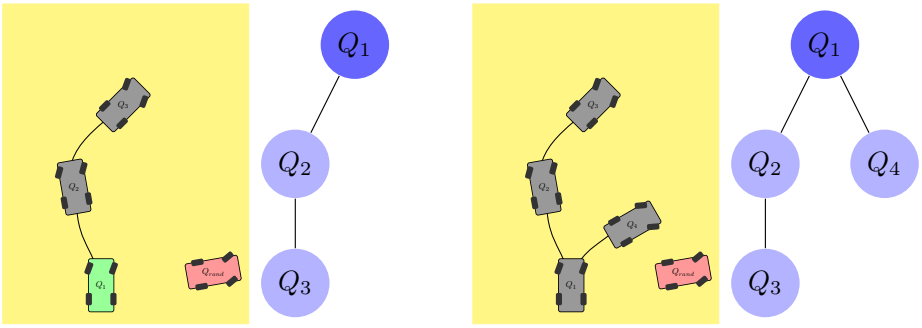
Figure 2.8



(a) Steering the nearest node in the direction of the random state

(b) Sampling a random vehicle configuration in the free space

Figure 2.9



(a) Finding the state in the tree nearest to the random vehicle configuration

(b) Steering the nearest node in the direction of the random state

Figure 2.10

2.2.2 Implementation

The details of how the previous procedures were implemented are now given. Several choices can be made when implementing the various steps in the algorithm. Whenever we deem necessary we will explain the reasoning behind the choices made.

Sampling

Function `RandomState` should return a random configuration. As discussed in [17], the distribution of this random configuration should be uniform along the free space. In the case of rectangular workspaces, one can simply create a distribution that returns the configuration $(x_{rand}, y_{rand}, \theta_{0_{rand}}, \theta_{1_{rand}})$ with $x_{rand} \sim \mathcal{U}(X_{min}, X_{max})$,$y_{rand} \sim \mathcal{U}(Y_{min}, Y_{max})$, and $\theta_{0_{rand}}, \theta_{1_{rand}} \sim \mathcal{U}(0, 2\pi)$. X_{min},X_{max},Y_{min} and Y_{max} represent the boundaries of the XY plane in which the truck is moving.

The configuration that results from this random sampling is then checked with an obstacle collision checker (described in detail below), and if it is a valid configuration, will be returned, otherwise the algorithm will simply continue on trying new random configurations until one of them is valid.

When `RandomState` is called, with a probability of $p_{goal} = \frac{1}{20}$ the goal configuration will be returned as the random sample. Otherwise it will simply run the previous procedure and return a random configuration. The goal configuration is the center of the goal region, and it is changed depending on the metric being used, more details are given in 2.2.2.

The probability p_{goal} helps define the greediness of the algorithm. A high value of p_{goal} will make the search try to converge to the goal more often, thus making it greedier. In highly constrained environments a greedy algortihm will perform worse, because it will sacrifice the ability of exploring the free space, by trying to converge to the goal more often, this can result in the algorithm becoming traped in difficul areas of the environment. The decision

to set $p_{goal} = \frac{1}{20}$ was based on the implementation described in [18], and also on a good performance resulting from the choice of this value.

Collision Checking

The collision checking procedure is often executed in various sections of the algorithm. As such, one should be concerned with its computational efficiency. Since it is deeply connected with the representation of the environment, one must also consider a representation of the workspace that can work smoothly with the collision checking.

The truck and trailer system can be well approximated by two rectangles, with the dimensions corresponding to the truck and the trailer.

To check if a given configuration $(x, y, \theta_0, \theta_1)$ is in collision with an obstacle we can simply check if any of the polygons corresponding to the vehicle collides with any obstacle. By representing the obstacles as polygons, we can simply check if there are any polygons corresponding to the vehicle intersecting with any of the polygons corresponding to the obstacle.

The intersection function is implemented using MATLAB's `polybool` function with the flag `'intersection'`. Using this function one can know if there is any kind of intersection, and thus a collision.

Nearest Neighbour

Function `NearestNeighbour` finds the closest node in the tree to a given x_{rand}. The nearest neighbour is found by exhaustively searching the whole tree $\mathcal{T}$.

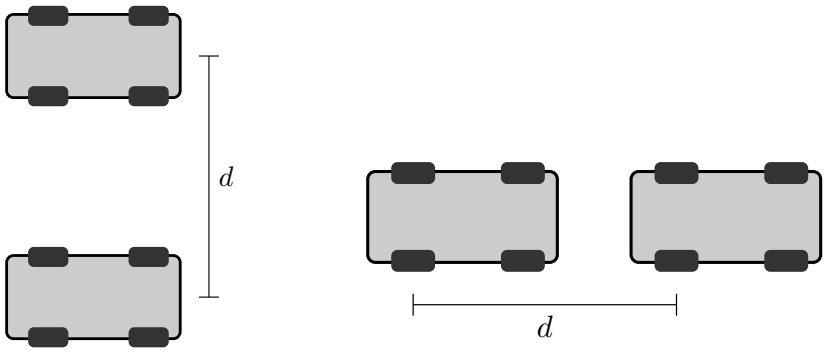
One can define many distance functions besides the Euclidean distance (complex metric examples can be found in [19] and [20]). The choice can have a profound impact on the performance of the algorithm [21], and is usually the result of a heuristic thinking.

A difficulty arising with the simple euclidean distance is the comparison of different units, figure 2.11 shows a possible problem.

In the figure we can see two situation in which the euclidean distance yields the same value. However one should notice that figure 2.11a represents a situation in which is much harder to reach the goal, as compared to figure 2.11b in which the car simply has to move forward a distance of d.

Finding the perfect heuristic is as hard as finding the exact solution to the planning problem. Being so, one can only try to experiment different distance measures and keep the best. Finding good heuristics is a problem often addressed in the literature.

During our experiments we used a distance measure based on the XY euclidean distance from the front of the truck, to the point at which we wish to arrive. The idea behind our rationale is that the closest node should be the one which corresponds to having the truck heading closest to the objective state. The orientations (θ_0, θ_1) are here ignored, for they are of little use when



(a) Euclidean distance of d (b) Euclidean distance of d

Figure 2.11

the truck is at distances much larger that $2 \times \pi$, corresponding to twice the maximum possible distance metric of each angle, π.

Steering

The `Steering` function is responsible for finding a path from the nearest state to the sampled state x_{rand}. One can choose between trying to find a path that will get as close as possible to the sampled state, or to simply find a path of a fixed length that will get us closer to the sampled state. The former method is defined as Connect and the latter as Extend. As discussed in [18] and [22], the Connect method usually yields worse results in nonholonomic problems, due to its greedy characteristic. As such we will choose to use the Extend method.

In the extend process we will limit the travelled distance by setting the input velocity command v to a fixed value. As a rule of thumb we define v to be dependent on the dimension of the workspace. It is defined to be proportional to the dimension of the workspace, thus resulting in larger steps in a larger environment, and more precise movements in smaller environments.

We discretize the allowed steering angles to an interval of three values corresponding to $[-\phi_{max}, 0, \phi_{max}]$, a more fine discretization would provide better results, however it would also increase the computational effort. For each steering input we simulate the system for one second. The simulation is done by performing Euler integration on the vehicle kinematic model. For the integration a time step of 0.1 seconds is used, thus resulting in 9 intermediate states. For each of these states we check if they are valid, by performing a collision check and a jackknife check.

After we have the three possible end states, corresponding to the three possible steering angles, we choose the one which has the smallest distance measure to x_{rand}. This new state is added to the tree, alongside with the steering input used.

Goal Checking

The goal checking procedure is a simple one, with objective of determining if a state as arrived at the goal region. The goal region is defined as a circle centered at an (x_g, y_g) position, with a radius r_g. In order to check if a state is at the goal region we simply compute the distance of the state to (x_g, y_g) using the defined metric. If this distance is smaller than r_g we consider that the state ar achieved the goal region.

If the metric being used is also considering the orientation states θ_0 and θ_1 then we must also define a goal orientation θ_{0g} and θ_{1g}, the metric distance is then computed and we check if it smaller than a certain tolerance ϵ. It is important to notice that we are not defining the goal as a region, but instead as a neighbourhood close to $(x_g, y_g, \theta_{0g}, \theta_{1g})$.

2.2.3 Results

Here we present some of the problems solved using the RRT algorithm. Figure 2.12 shows two different environments, characterized by being very constrained, *i.e.*, with very tight spots for the truck to drive in. In blue, one can see the paths expanded by the tree. These paths, correspond to the movement of the truck pulling the trailer. The path in red consists in the goal solution found, and it represents the movement of the trailer (not to be confused with the movement of the truck).

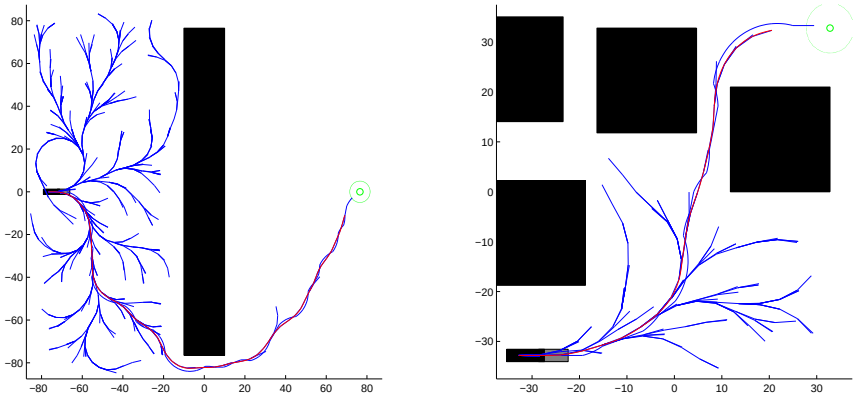


Figure 2.12: Solving heavily constrained environments

Figure 2.13 shows environments that are not as constrained as the previous ones, but that are much larger.

All of the shown environments were solved with relative ease, proving that the algorithm is suited for this particular situations.

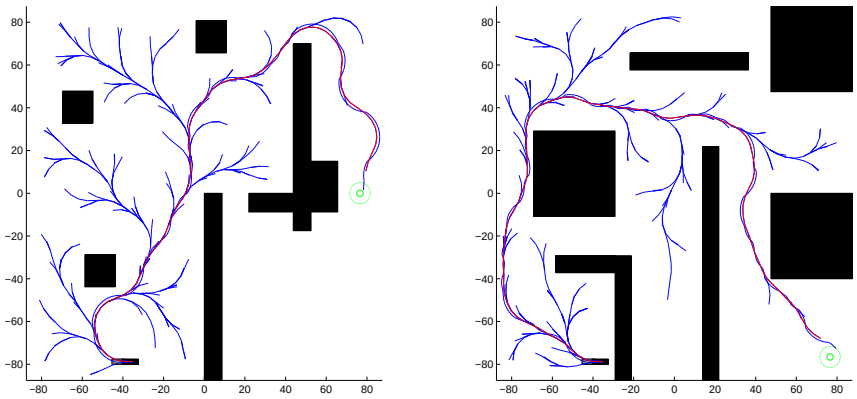


Figure 2.13: Solving large environments

2.3 Reducing RRT's Metric Sensitivity

2.3.1 The algorithm

As stated previously the RRT is sensitive to the chosen metric, and as such can behave poorly if the used metric does not truly reflect the real distance to the goal. In [23] the original formulation of the RRT is adapted, in an attempt to reduce the influence of the chosen metric in the final result.

This change is done by adding two new functionalities, which will be called Exploration Information and Path Collision Tendency.

Exploration Information

The exploration information consists in keeping additional information in each node in the tree. This information keeps track of which inputs were already tried for a determined state.

Every time a node is chosen, and the possible children are attempted (resulting from the possible steering angles $[-\phi_{max}, 0, \phi_{max}]$), a record is stored, containing which inputs already resulted in a child node and which resulted in collision.

For each iteration of the RRT, when the nearest node is picked, only the inputs that did not originate already a child or are not in collision are tried. In the case that all of the inputs become exhausted (either by generating children or by being in collision), the node is removed from the tree, and excluded from future nearest-searches.

The objective of this new functionality is to make the algorithm more efficient in exploring the environment, and also remove potential nodes that can cause

the algorithm to be trapped (by being constantly selected as the nearest node and at the same time having all of its children in collision).

Path Collision Tendency

The second added functionality consists in keeping track of a collision tendency. Before a new child node is added to the tree, it is first tested for its collision tendency. One does so by checking how many of its possible children will be in collision/jackknife. The number of descendants in collision over the number of possible descendants, yields the probability of future collision of the parent node.

$$p_{collision} = \frac{\#children_{collision}}{\#children}$$

Figure 2.14 illustrates the computation of the probability of collision.

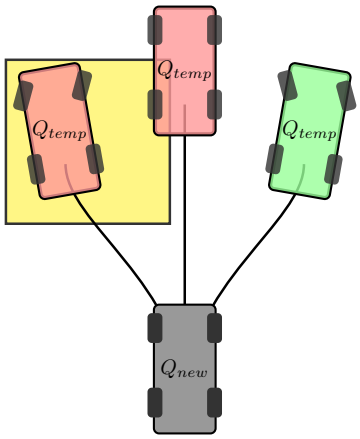


Figure 2.14: The probability of collision of a new node is equal to the number of children nodes in collision over the number of possible children node, in this case $^2/_3$

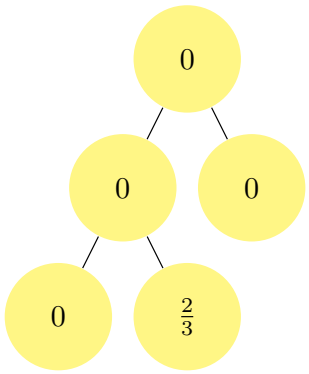
Whenever a new node is added, this information is also stored alongside with it. An additional step is made which consists of propagating this probability to the parent of the new node. The influence the new probability of collision of the child has on the parent, is given as

$$p^{parent}_{collision} = \frac{\#children_{collision}}{(\#children)^2}$$

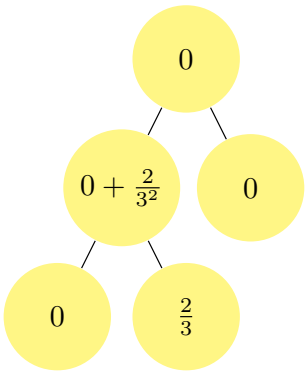
This process is continuously propagated to all of the predecessors of the original node. For each level that the propagation goes up, the probability decreases by a factor of $\#children$, thus the i^{th} parent of the new node will have an additional probability collision of

$$p^{i}_{collision} = \frac{\#children_{collision}}{(\#children)^{i}}$$

Figures 2.15 and 2.16 show an example of propagating the probability of collision from a newly added node until the root node.

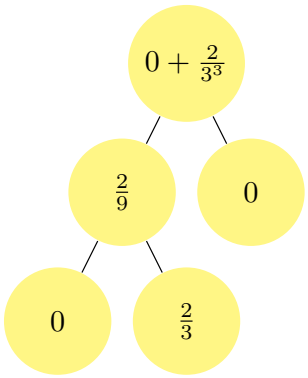


(a) Probability collision of newly added node

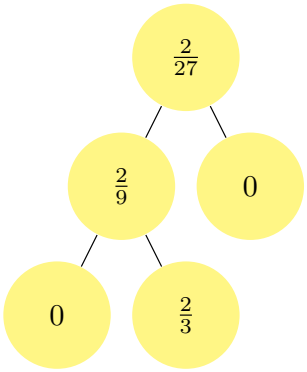


(b) Propagating to the first parent

Figure 2.15



(a) Propagating to the parent of order 2



(b) Final path collision tendency tree

Figure 2.16

By propagating the collision probability of a new node to its parents, one can get a lower bound on the true probability collision of each node in the tree. This information is of value, since it gives an estimate on the collision probability of the nodes, without too much computational effort. Also, we know that if one given node has a high collision probability, then one might expect a high probability of its descendants being in collision.

We can make use of this collision probability information in order to improve the search. Whenever a node in the tree is chosen as the nearest, with a probability $p_{discard}$ equal to its stored collision probability, we simply don't try to expand the node, and jump to the next iteration.

In this way we are trying to avoid nodes that present a high probability of collision, and effectively cropping the tree growth into dangerous regions.

2.3.2 Implementation

To store the exploration information associated with each node, three additional elements are added to the node structure. The first one is a vector which keeps track of the already tried inputs and the ones that result in collision, in any of these situations the element corresponding to the input is set to 1, otherwise it will remain with a value 0. A second variable keeps track of the collision probability computed for each node.

The third variable contains the identification of the node's parent. By having this information associated to each node, it becomes very simple to propagate the collision probability from a given state up to the root node.

2.3.3 Results

In order to evaluate the performance this second algorithm, a comparison is made with the original RRT.

The problem to be solved here corresponds to an environment with a grid of obstacles, as seen in figure 2.17.

The truck is situated on the south west region of the map, with an initial orientation of π. The goal region is located on the north east region, inside the obstacle grid.

Three different metrics are used for as measurements of nearness, they are as follows

Euclidean distance
$d(X_a, X_b) = \|(x_a, y_a, \theta_{0_a}, \theta_{1_a}) - (x_b, y_b, \theta_{0_b}, \theta_{1_b})\|$

Euclidean xy distance
$d(X_a, X_b) = \|(x_a, y_a) - (x_b, y_b)\|$

Truck heading distance
$d(X_a, X_b) = \|(x_{h_a}, y_{h_a}) - (x_{h_b}, y_{h_b})\|$, where the subscript (x_h, y_h) stands for the position of the frontal bumper of the truck, in a truck and trailer system.

The objective of this second version of the RRT algorithm is to reduce the sensitivity of the algorithm to the chosen metric, thus achieving better results than its original version, even if a poor metric is chosen.

To test these results we run the same instance of the problem 1000 times for each metric, and for both the original RRT algorithm and the improved version. The results of these runs are in table 2.1.

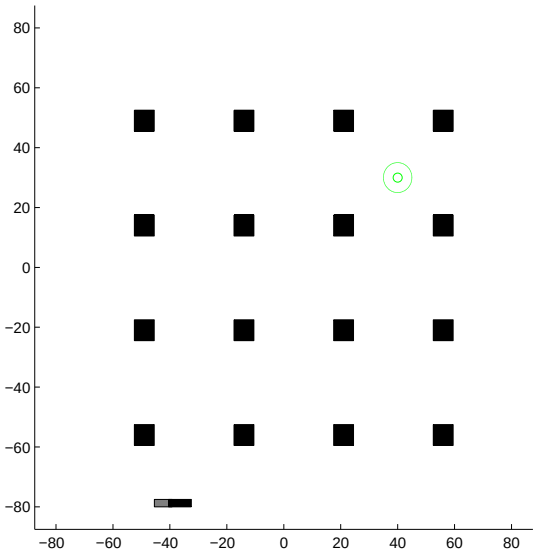


Figure 2.17: Problem instance to be solved

Metric	Path distance (average)		Nodes expanded (average)	
	RRT	RRT Improved	RRT	RRT Improved
1	246	238	1031	639
2	246	239	787	532
3	206	206	402	408

Table 2.1: The average path distance of the found solutions of the RRT and RRT improved

By observing table 2.1 we can see that for metrics 1 and 2, the RRT Improved algorithm, finds, on average, a significantly shorter solution. For the third metric however, the path distance is the same.

In terms of the nodes expanded while the search for a solution is done, we notice also an improvement for metrics 1 and 2, where the RRT Improved expands less nodes than the RRT. For the case metric 3 however, there is a slightly larger number of nodes explored by the RRT Improved.

The results above show that the RRT Improved algorithm is usually able to find solutions with roughly the same, or a higher quality (path length) than the RRT. This improvement is specially noticeable when the metrics used are of a poorer quality.

2.4 RRT*

The RRT* algorithm was developed to address the main drawback of the RRT algorithm, its sub-optimality. The original implementation of the algorithm was proven to converge to the optimal solution as time tends to ∞. The algorithm is fundamentally similar to the RRT, however it keeps on improving its solution even after one is found.

2.4.1 The algorithm

The algorithm in its original formulation can be found in [24]. We will show here how we adapted the algorithm to our particular problem.

This algorithm requires a few changes and the addition of a set of new functionalities.

We will now have the need for a distance measure $\texttt{D}(x_a, x_b)$, which is given by the length of the path found between states x_a and x_b.

Each node in the tree will also keep record of a cost `C`, which is equal to the sum of all path lengths from itself to the root node. This cost is the simple addition of all distances `D` between the node and its parent, its parent and its parent's parent, and so on, until the root node is achieved.

We must also include two new functions `NearFrom` and `NearTo`, which when given a state x, will find the closest nodes in the tree which can be achieved from x, and which can achieve x, respectively.

Finally, one must also add a new tree management function, `RemLink`, which will disconnect a node from its parent. The function $\texttt{AddNode}(x_a, x_b)$, will work as before, adding the state x_b as a child of x_a.

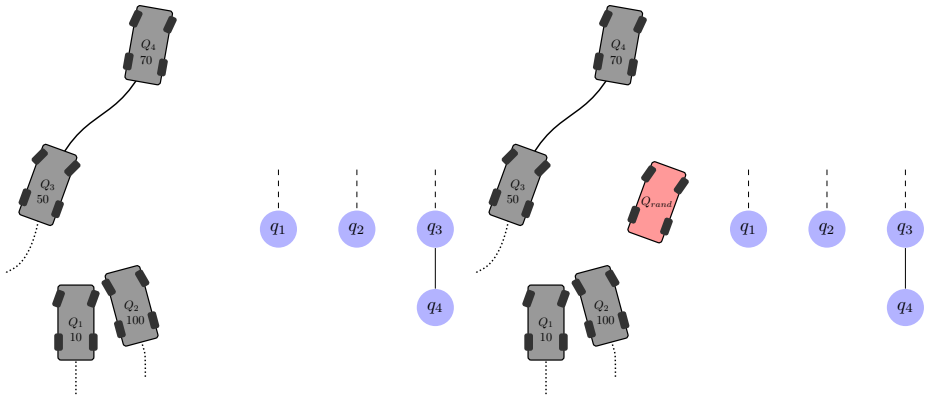
The algorithm is similar to the original RRT expressed in 1. In algorithm 2 we show the difference in the main `while` loop, everything previous to the `Steer` procedure is exactly the same as the original RRT.

A visual example of how the algorithm works is shown in figures 2.18 to 2.21. In each image, we can see the tree structure in the right. The tree is supposedly in an already complex configuration, and we are only looking at a portion of it. On the left we see the visual representation of the tree states and

$(u, x_{new}) \leftarrow$ `Steer`(x_{rand}, x_{near});
$X_{near} \leftarrow$ `NearTo`$(x_{new}, \mathcal{T})$;
$x_{min} \leftarrow x_{nearest}; c_{min} \leftarrow$ `C` $(x_{nearest})+$ `D` $(x_{nearest}, x_{new})$;
foreach $x_{near} \in X_{near}$ **do**
  if `ObstacleFree`$(x_{near}, x_{new}) \wedge$ `C` $(x_{near})+$ `D` $(x_{near}, x_{new}) < c_{min}$ **then**
    $x_{min} \leftarrow x_{near}; c_{min} \leftarrow$ `C` $(x_{near})+$ `D` (x_{near}, x_{new})
end
$\mathcal{T} \leftarrow$ `AddNode`(x_{min}, x_{new});
$X_{near} \leftarrow$ `NearFrom`$(x_{new}, \mathcal{T})$;
foreach $x_{near} \in X_{near}$ **do**
  if `ObstacleFree`$(x_{new}, x_{near}) \wedge$ `C` $(x_{new})+$ `D` $(x_{new}, x_{near}) <$ `C` (x_{near}) **then**
    $x_{min} \leftarrow x_{near}; c_{min} \leftarrow$ `C` $(x_{near})+$ `D` (x_{near}, x_{new})
end
$\mathcal{T} \leftarrow$ `RemLink`(x_{near});
$\mathcal{T} \leftarrow$ `AddNode`(x_{new}, x_{near});

Algorithm 2: RRT* algorithm

its connections. Associated to each state is a number (*e.g.* 120 for state Q_{104}), which represents the cost associated to the node.



(a) The current tree configuration (b) Sampling a random state

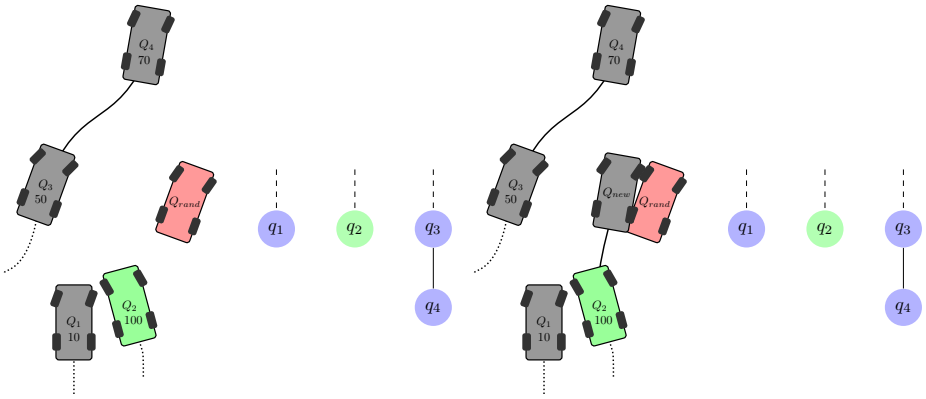
Figure 2.18

The previously shown process shows how the RRT* algorithm is able to improve the solutions encoded by the tree. One can intuitively expect that if this process is done often enough, and with a random sampling of the new states, then one will find the optimal path.

We do not prove that the solution will tend to the optimal, but in [25] one can find that proof for simpler systems.

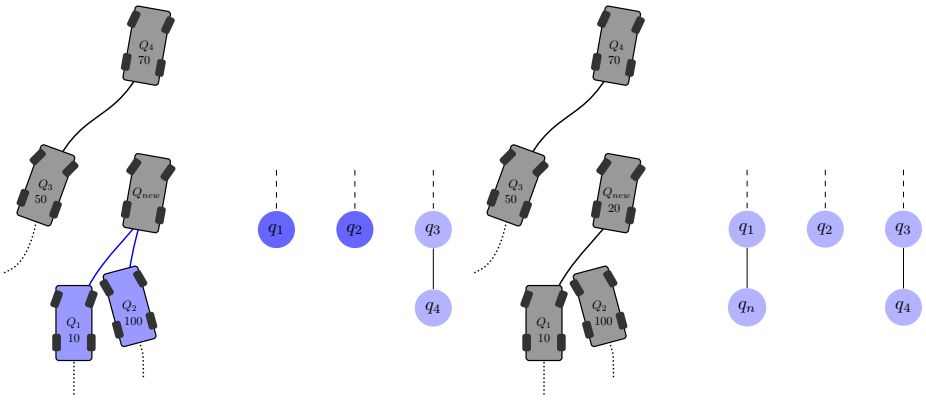
2.4.2 Implementation

The realization of some of the procedures mentioned before is not obvious, as such, we will detail how they were implemented.



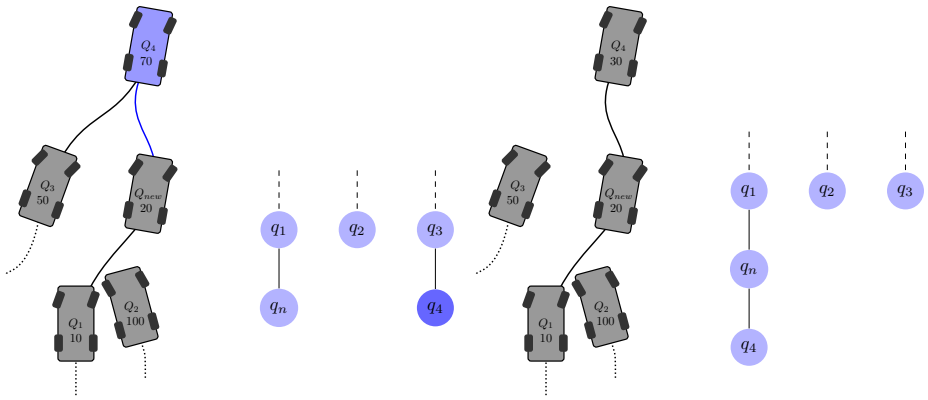
(a) Finding the nearest state in the tree (b) Steering towards the random state

Figure 2.19



(a) Finding the near nodes which can achieve Q_{new} (b) Choosing the parent which yields the minimum `C` + `D`

Figure 2.20



(a) Finding the near nodes which can be achieved from Q_{new}

(b) Rewiring near nodes if it results in a smaller `C + D`

Figure 2.21

Steer

The function `Steer` used for the RRT* is different from the one used in the previous algorithms.

The RRT* requires that some states be exactly connected, for example when we are trying to connect a state to its near neighbours (resulting from `NearFrom` or `NearTo`). In order to do this we must abandon the approach of trying the discretized values of the steering angle $[-\phi_{max}, 0, \phi_{max}]$, for they will most likely not be able to connect two different states.

The alternative is to use Dubins paths [12]. A Dubins path is the optimal (shortest) path between two car states, in the absence of obstacles. It can be computed easily, since it is always a simple combination of straight lines and segments of arc.

For the computation of Dubins paths we used the publicly available code found in [26].

When trying to steer the current state to a desired state, the `Steer` function will compute the Dubins path that connects both points. The resulting state from `Steer` can either be the desired state, or an intermediate point, corresponding to how far in the path can one go without exceeding the maximum allowed travelling distance $dist_{max}$. If there is any collision in the path corresponding to the initial $dist_{max}$ meters the solution returns a failure.

Near To

As opposed to the original formulation from Sertac our system is not equally steerable in any direction. This happens because we are not allowing the truck to move backwards, and thus we effectively reduce the nearness symmetry that the truck might have in the forward and backward direction. Figure 2.23 illustrates

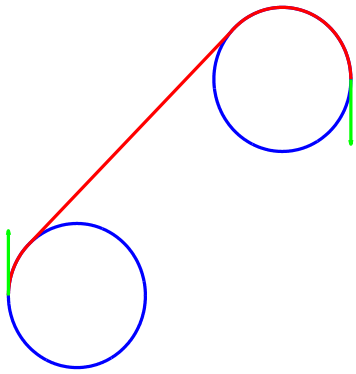


Figure 2.22: An example of a Dubins path starting from the left green arrow,enconding a state (x, y, θ), and ending in the right green arrow)

this.

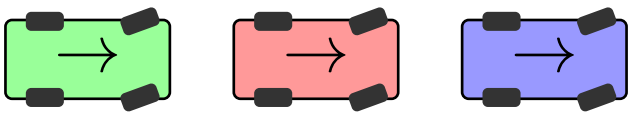


Figure 2.23: The asymmetry in the nearness function. Due to not being able to reverse, the red truck is at a bigger "effective" distance to the green truck than to the blue truck

From this problem arises the need for developing two separate nearness functions `NearTo` and `NearFrom`. The function `NearTo` is responsible for finding the nodes that are near a certain state and that can achieve that state. In order to achieve that, we look for states situated in the back of our desired state (remember that the truck states can only move forward). So we look for tree nodes located in a cone centred at the truck rear, and pointing backwards. The cone has an angle of $2 \times 45°$ and a radius of r_{max}.

To further reduce the possible near states, we also require the angle of the truck states to have a maximum difference of 45° to the desired state. Figure 2.24 shows the selection of near states respecting all the previously mentioned conditions.

The previous figure illustrates the effectiveness of finding possible near nodes, by applying a simple set of constraints. It represents a big advantage to do this pre processing of possible neighbours, for to test Dubins paths with all the nodes in the tree would require a large amount of time.

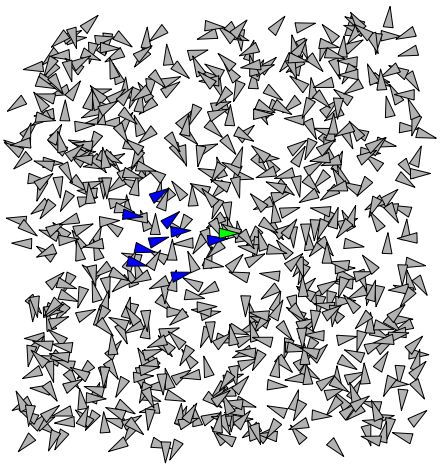


Figure 2.24: The blue triangles are nodes that might achieve the green triangle

Near From

The `NearFrom` is very similar to the `NearTo` procedure. The only difference is the direction of the cone. Instead of looking for the states in a cone pointed backwards and situated on the rear, we look for states in a cone pointed forward and situated on the front. By doing so we effectively only select nodes that are possible to achieve from the current state, as figure 2.25 shows.

2.4.3 Results

Figures 2.26 to 2.28 show the RRT* solving a possible problem configuration. These figures illustrate the tree growth and how the path is successively improved. A drastic change is even observed when, during execution, the algorithm changes its best path from one on top of the map, to one in the bottom (figure 2.27).

The algorithm continues to expand the tree, whilst improving the found solution path. The path shown in red, in the figures corresponds to the path leading to the best node in the tree, which is the one with the minimum value of the sum of distance to goal plus travelled distance (the node is also required to be in a vicinity of the goal region).

A note must be made on the performance of the RRT* algorithm. In our situation, we were not able to obtain satisfying results from the RRT*. The reason being that a high computational effort is needed to run the algorithm, thus

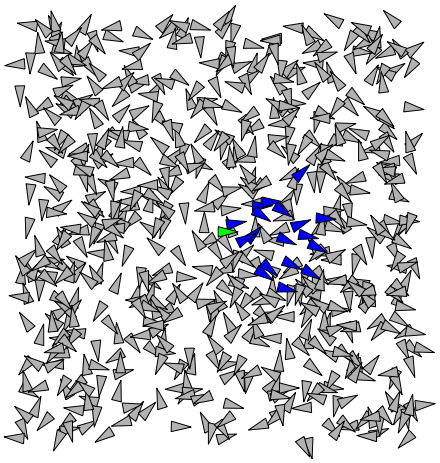
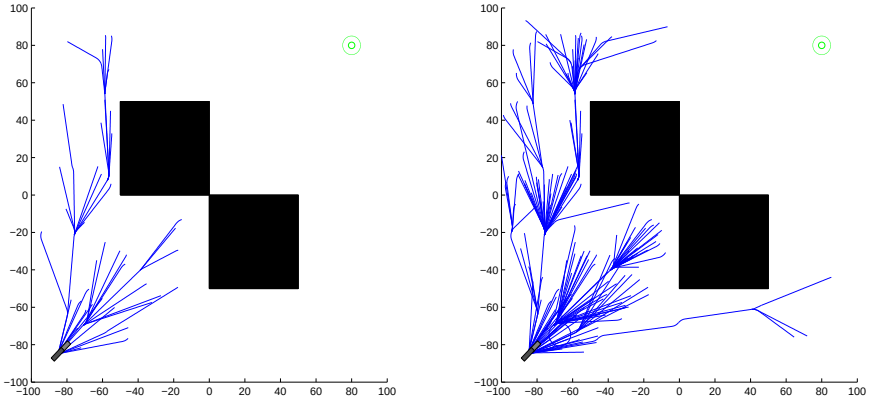
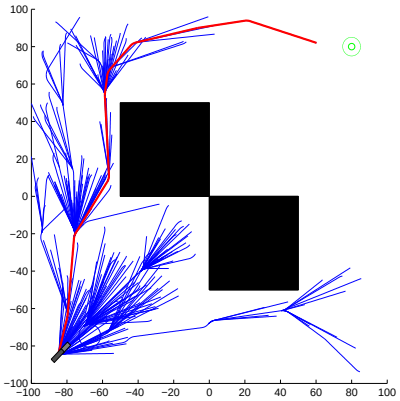


Figure 2.25: The blue triangles are nodes that might be achieved by the green triangle

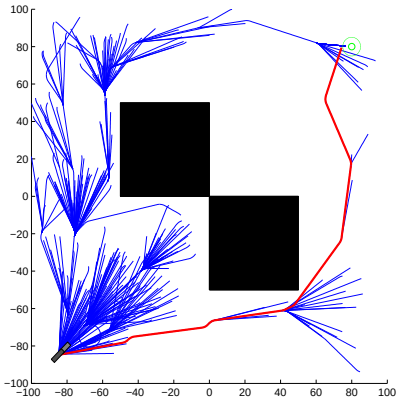


(a) Intermediate iteration of RRT*-1 (b) Intermediate iteration of RRT*-2

Figure 2.26

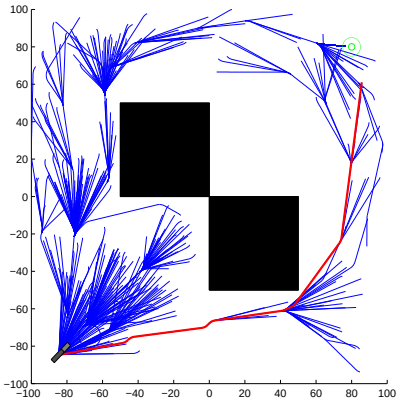


(a) Intermediate iteration of RRT*-3

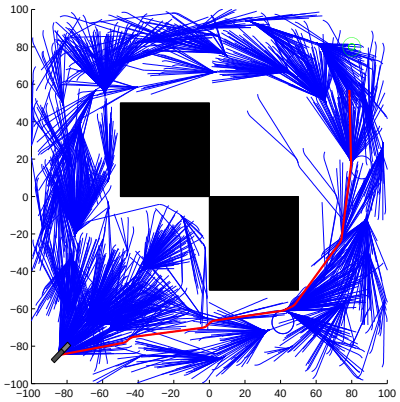


(b) Intermediate iteration of RRT*-4

Figure 2.27



(a) Intermediate iteration of RRT*-5



(b) Intermediate iteration of RRT*-6

Figure 2.28

making it impractical to use. This high complexity is related to the difficulty in growing and connecting nodes in the tree, which requires a large number of conditions to be met.

2.5 Path Optimization

As seen before the RRT algorithms result in far from optimal solutions. Except for the case of the RRT* algorithm, which converges to the optimal solution, one expects the other RRT algorithms to result in highly sub optimal solutions. A common approach to improve sub optimal path solutions often consists in simplifying the original path in order to remove its unnecessary segments [27], we will however follow a different approach.

We can try to soften this problem by performing a post RRT local optimization. The objective of this optimization is to shorten the path, by trying new paths that are variations of the one found by the RRT. One of the big improvements that this post processing achieves is a much smoother path as will become obvious in the following pages.

2.5.1 The algorithm

The developed optimization is based on the branch of Artificial Intelligence known as Genetic Algorithms. For an extensive survey on the topic refer to [28].

The output of the RRT comes in the form of a sequence of tree nodes, corresponding to the path in tree, that takes us from the root node until the node that achieved the goal. Similarly each node corresponds to a state which is a part of the path to the goal.

Knowing that the RRT usually results in paths longer than necessary, one can ask if all the intermediate nodes that lead to the goal are really needed to define the path. Figure 2.29 shows an example of unnecessary nodes belonging to a found solution.

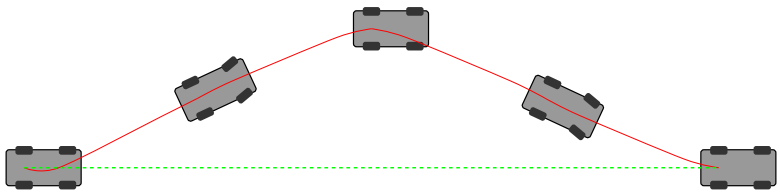


Figure 2.29: The original path is shown in red. A possible and very simple improvement of the path, consists in ignoring the nodes that represent an unnecessary route

Finding the unnecessary nodes however, is not an easy process. We will rely on genetic algorithms to perform that task.

A genetic algorithm will try to optimize a given cost functional, by changing its input, usually given by a string of bits.

As said before, a path output found by an RRT algorithm, will come in the form of a sequence of states, corresponding to nodes in the tree. The optimization variable in our genetic algorithm will be a string of bits, with a length equal to the number of states in the sequence. Each bit represents if the node is used in the path to the goal (value 1) or not (value 0).

Initially the string of bits will be of the form $11\ldots1$, corresponding to all of the nodes being used in the path to the goal. The genetic algorithm will then have to change the bits in the string, such that it finds the best combination of states which still achieves the goal.

One can measure the quality of a string through the objective function, which is defined to be the distance travelled in the path encoded by the string. The algorithm will seek to minimize this function, hopefully resulting in a better path.

2.5.2 Implementation

In this section we will give the details on how the genetic algorithm was implemented.

The string of bits

Noticing that both the initial and final nodes, corresponding to the initial and final configuration, will always have to be present in the optimized path, we actually reduce the string of bits to have a size equal to the number of states minus two (the initial and final states).

The objective function

The objective function should take as input the optimization variable and return a measure of how good the solution is.

When a bit string is received the corresponding active states (nodes with values of 1) are connected through Dubins paths. The corresponding path is checked for collisions and its total distance is computed. The distance will correspond to the output of the objective function, since it is this distance that we are trying to minimize. To avoid colliding solutions, whenever a collision is found the objective function will return an infinitely large distance, thus making the corresponding bit string unappealing for the genetic algorithm. In figure 2.30 an example of a possible string bit combination and its respective path is shown.

Genetic Algorithm Details

The initial population of the algorithm consists in two strings, one consisting of only ones, thus corresponding to the original RRT solution, and the second string consisting of only zeros, thus being an attempt to directly connect the starting state to the goal state with a Dubins path. The remaining elements of

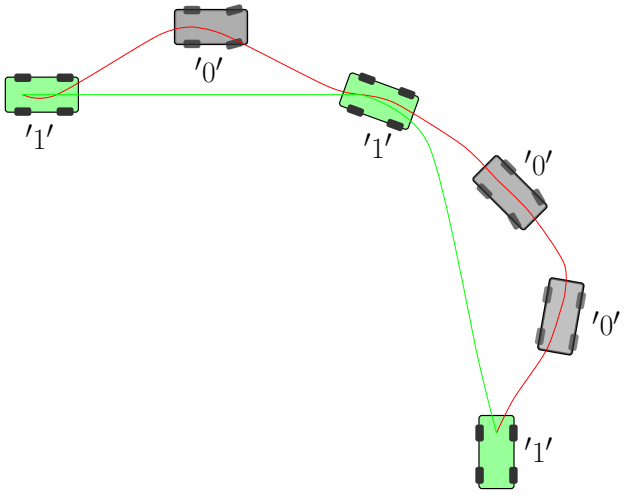


Figure 2.30: The path corresponding to a string bit of ′101001′

the population are randomly generated strings, where each bit is set to 1 with a probability p_1.

In order to always have a safe set of possible string combinations, we decide to propagate the 20% top ranked (smaller distance) individuals in the current population to the following population. The remaining 80% of individuals are obtained as explained below.

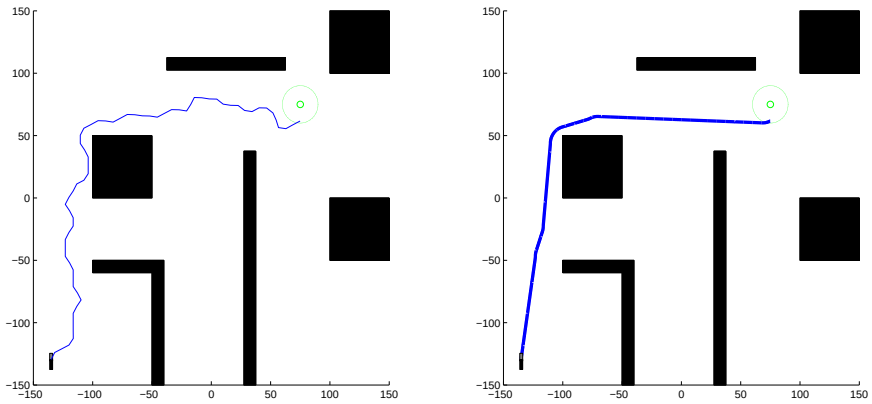
One of the fundamental operations in Genetic Algorithms is the crossover operation. The 1-point crossover was chosen, and it is implemented by randomly selecting a splitting region in the bit string. We then pick randomly two strings from the previous generation, there is some bias to pick the highest rated strings more often that the others. From each of those two strings we pick complementary regions divided at the splitting region, and perform a 1-point crossover.

If the crossover process creates an already existing individual in the population, that is, the algorithm starts to prematurely converge, we induce a mutation in the population, by creating a new individual, in the same fashion as the ones randomly created in the first population.

2.5.3 Results

In figure 2.31 the results of an RRT instance, and the subsequent local optimization are shown. For the optimization, a total of 5 generations, with 10 individuals each, were used.

The local optimization improved the quality of the found solution by making it shorter. However the biggest improvement comes from the smoothing of the path. The original RRT solution had very aggressive steering manoeuvres, which disappeared after the path optimization.



(a) Solution found by the RRT algorithm (b) Post processed RRT solution

Figure 2.31

Chapter 3

Truck Modelling

This chapter will focus on modelling the RC Trucks that will be used to test the controllers explained in Chapter 4.

The modelling process plays a crucial role in the implementation of a controller in a real world setting, for without a thorough knowledge of the system to be controlled, the implementation of a controller can result in failure, even though said controlled shows good results in theory.

Some of the problems that usually appear in real life settings, but are usually not considered are:

- Sensor noise
- Quantization
- Actuator saturation
- Actuator deadband
- Actuator delay
- Unmodelled dynamics

Throughout this chapter we will try to address some of these undesired characteristics, and use methods that attempt to remove their influence on controller performance. We will also explain the overall setting of the test bed environment.

3.1 The RC Truck

3.1.1 Introduction to the RC Truck

The RC truck to be used is a 1:32 scale model of the Scania R260 truck with a detachable trailer. Figure 3.1 shows an image from the on-line catalogue of the vendor SIKU.

Through the remote, the user can control a vast range of actions, besides only driving the vehicle.

Figure 3.1: A catalogue image of the truck to be used. Courtesy of: http://www.siku.de/

3.1.2 Truck dimensions

It is important to know the dimensions of the truck being used, since they are needed by some of the controllers discussed in 4 and the motion planning algorithms explained in Planning Chapter.

Figure 3.2a shows the measures taken for the longitudinal dimensions of the truck. These measures include the distance L_{truck}, between front and rear axles of the truck, distance $L_{trailer}$, between the middle wheel axle of the trailer and the hitch point. B_{front} and B_{back} represent bumper lengths, which are needed by the planning algorithms in order to check for collisions with obstacles. The measured lengths are $L_{truck} = 0.115m$, $L_{trailer} = 0.19m$, $B_{front} = 0.04m$ and $B_{back} = 0.08m$.

Figure 3.2b gives us the width of the truck B, which is roughly the same as the width of the trailer. This width is also required by the collision checking procedure of the planning algorithms.

3.1.3 Commanding the RC Truck

The remote was dismantled and connected to a National Instruments acquisition board. The connections are such that the acquisition board can set a desired voltage to each of the relevant remote control inputs (velocity and steering).

The acquisition board is connected to a computer, which can interface with it through MATLAB commands. The MATLAB commands can specify a desired

(a) Measured lenghts of the RC Truck (b) Measured width

Figure 3.2

voltage to be applied in each of the output channels of the acquisition board.

The RC Truck is commanded as follows, a MATLAB script interfaces with the National Instruments board, issuing desired voltages to it, which are then applied to the remote control, finally resulting in a reaction by the truck. This chain of events is depicted in figure 3.3.

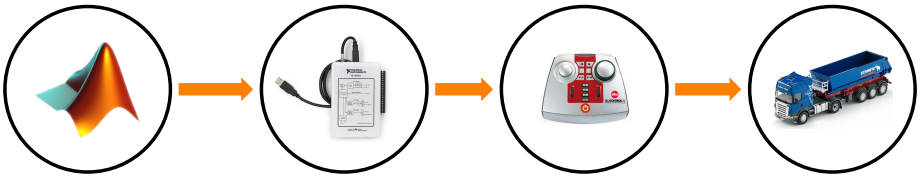


Figure 3.3: A representation of the chain of events since a command is issued in MATLAB

3.2 Sensing the Truck

3.2.1 Qualisys motion capture system

The RC Truck does not have any sort of available odometry. As such, it is impossible to directly retrieve any kind of information from the truck alone.

The test bed environment is equipped a Qualisys motion capture system, consisting of 12 cameras (as seen in figure 3.4a) mounted in strategic locations of the environment. These cameras can retrieve an accurate pose estimate of an object equipped with a set of special reflective markers (figure 3.4b).

For our purposes, we are only interested in the x and y coordinates, and the yaw (equivalent to θ) orientation.

(a) A catalogue image of a camera.
Courtesy of: http://www.qualisys.com/

(b) A catalogue image of the used markers.
Courtesy of: http://www.qualisys.com/

Figure 3.4

The Qualisys system is interfaced with MATLAB commands, and can provide the current pose of an object, when asked for it. The truck and the trailer are set up as different objects in the Qualisys software (it is important to remember that even though the truck and the trailer are attached to each other, their yaw orientation is different, thus needing to estimate the yaw orientaion of two different objects).

Using this system we have access to the information of the truck and trailer poses in MATLAB, in real time. A representation of the system is shown in figure 3.5.

Figure 3.5: Reading truck and trailer poses in MATLAB

3.2.2 Pose Estimation

As explained previously the Qualisys system can provide us with a pose estimate for the truck and the trailer. In order to assess the reliability of these estimates, the following experiment was performed:

> *The truck has been placed in the workspace, and it is kept static. The Qualisys pose measurements are recorded, each measurement being taken at a 10 Hz rate. This experiment ran for two hours, yielding* $2 \times 60 \times 60 \times 10$ *pose estimates.*

In the obtained results we noticed the existence of noise. We modelled this noise, in a pessimistic way, through Gaussian distributions. The parameters found for these gaussians are presented in table 3.1.

	μ	σ
$\bar{x}$	x	0.0001 m
$\bar{y}$	y	0.00011 m
$\bar{\theta}$	θ	0.001 m

Table 3.1: Parameters of the Gaussian approximations

3.2.3 Velocity Estimation

Velocity estimates are needed for several reasons. They might be needed by control algorithms, or even to estimate the steering angle ($\phi = \arctan(\frac{L}{V}\omega)$) otherwise unmeasurable. In this subsection we explain the problems found and solutions used for the estimation of velocities.

Jitter

In some algorithms, like a control algorithm, it is important to maintain a fixed running time. This time is usually associated to the iteration of a `for` or `while` loop that is supposed to run several occurrences. Let us imagine algorithm 3.

```
initialization;
while true do
    start timer;
    do something;
    if timer>T then
        go to next iteration;
    else
        wait T−timer;
        go to next iteration;
    end
end
```

Algorithm 3: Maintaining a fixed running time of each iteration

In the previous algorithm, we can see that each iteration of the `while` loop is supposed to take T units of time. This however can fail if the `do something` section takes too much time to run. In that case, the running time of the iteration will be bigger than T. Assuming that the `do something` section actually takes less time than T to run, other imprecisions in the timing tools of the code, or in the `wait` function, or even in the overhead computations of the `while`, might result in slightly different running times of each iteration.

Jitter is defined as the phenomenon of execution time deviations of the control loop from its desired value of T. These variations can even induce instability in an otherwise stable control algorithm.

Besides the control considerations, the jitter can also influence the velocity estimates. The velocity along the x-axis, is estimated using the general discretization of the derivative:

$$\dot{x}_k = \frac{x_k - x_{k-1}}{T} \tag{3.1}$$

Where each sample $x(k)$ is taken at equispaced intervals of time T, *i.e.*, $x_k = x(kT)$.

If for some reason the algorithm is affected by jitter, the velocity estimate of x will be wrong, since we we are still using equation (3.1), when the interval of time between x_{k-1} and x_k is actually different than T.

Measuring the Jitter

When MATLAB requests the pose estimate to the Qualisys system, besides getting the (x, y, θ) values, it also gets an associated time stamp. This time stamp corresponds to the instant of time associated when the measurement was done. Using this information we can measure the jitter effect.

Algorithm 4 is run on a MATLAB script. The controller function is a dummy function that takes significantly less time than $T_{sampling}$ to run. The desired $T_{sampling}$ for the algorithm will be $0.1s$.

```
initialization;
k = 1;
while k < 100 do
    start timer;
    t_k=Qualisys time stamp;
    controller function;
    k = k + 1;
    if timer> 0.1 then
        go to next iteration;
    else
        wait 0.1-timer;
        go to next iteration;
    end
end
```

Algorithm 4: Measuring the jitter effect

The sampling time $T_{sampling} = 0.1$ cannot always be guaranteed to be respected, due to the reasons explained previously, resulting in jitter. By retrieving the time stamp associated to each Qualisys measurement we can get

a faithful measurement of the time each loop is taking. In Figure 3.6 is represented the time each `for` loop took to run, which is simply calculated as the difference of consecutive time stamps, $t_k - tk - 1$.

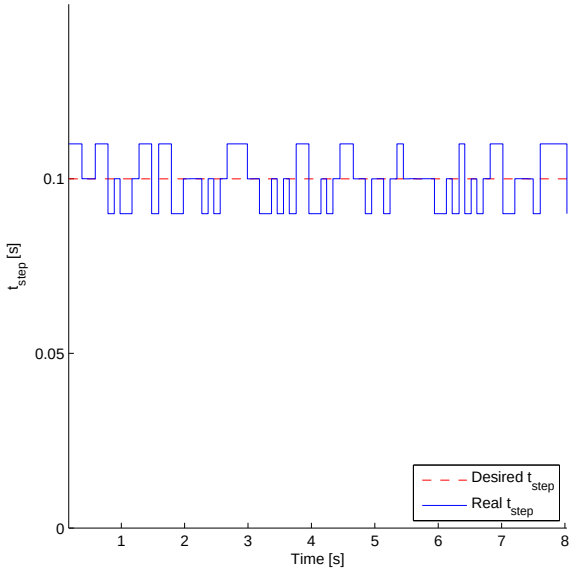


Figure 3.6: The jitter effect

Figure 3.6 illustrates the difficulty in maintaining a constant sampling time. Even though the real sampling time is around the desired sampling time, it oscillates slightly around the ideal value. Therefore it is important to take into account these small variations in the sampling time, and use the real sampling times (as obtained from the time stamps of the Qualisys sensor) in order to have reliable velocity estimates. Equation (3.1) is then changed to:

$$\dot{x}_k = \frac{x_k - x_{k-1}}{timestamp_k - timestamp_{k-1}} \tag{3.2}$$

Which should result in more accurate velocity estimates.

Linear Velocity

In this experiment the setup is as follows:

The truck is placed in the workspace. It is initially stopped. A voltage is applied as the velocity command. The steering of the truck is such that it moves (almost) in a straight line. The voltages applied to the velocity command are recorded at each time step, as are the instantaneous poses of the truck, given by the Qualisys system.

The velocity of the truck can be approximated by the following formula:

$$\frac{\sqrt{(x(t) - x(t - t_{step}))^2 + (y(t) - y(t - t_{step}))^2}}{t_{step}} \tag{3.3}$$

Where t_{step} is the real sampling time of the current iteration loop, as explained previously. The voltage wave in figure 3.7 is applied as the velocity command, and the measured velocity is shown in figure 3.8a.

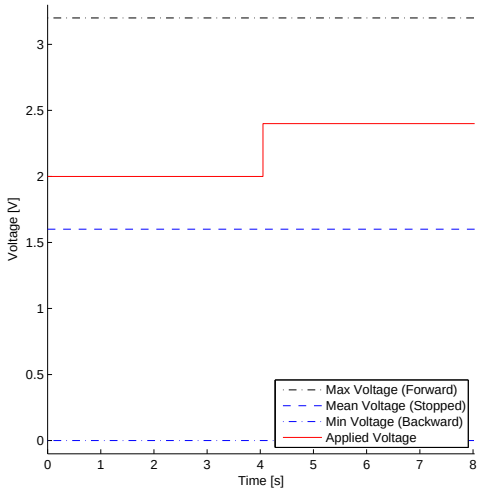
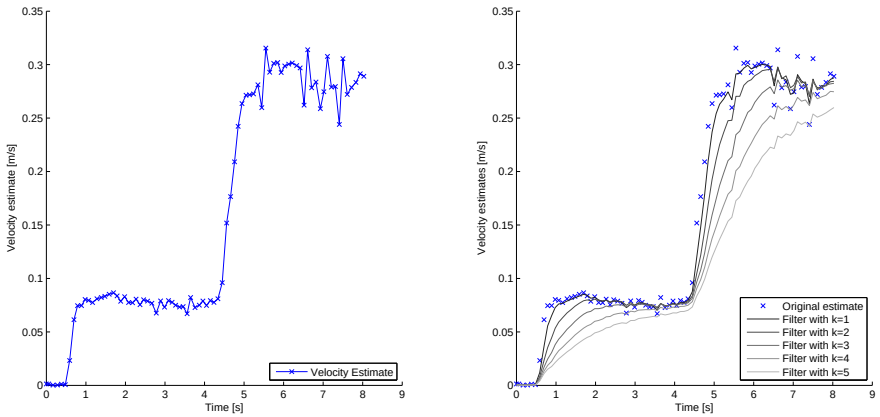


Figure 3.7: Voltage applied as velocity command

As we can see the measured velocity is affected by noise. This effect only gets worse as the sampling time t_{step} gets smaller. This prevents us from using very low sampling times, therefore, the decided sampling time was set on the previously mentioned $0.1s$, as a result of a trade-off between control velocity (control sampling time) and estimation quality.

The velocity estimate can be enhanced by using filters to try and remove noise. A low pass-like filter can be applied, by computing the current velocity estimate as a function of the previous velocity estimates:

$$v^*(n) = \frac{\sum_{i=1}^{k} v^*(n-i) + v(n)}{k+1}$$



(a) The measured velocity, when applying the voltage commands from figure 3.7

(b) Low pass-like filter, with different k values

Figure 3.8

Where $v^*(n)$ would be our improved estimate, and $v(n)$ would be our initial estimate, as computed from (3.3). Figure 3.8b shows the results of this filter for different values of k.

From the figure, we conclude that the bigger the k the smoother the estimate, however the estimates suffers from a delay, which only increases with k, Therefore a smaller k is preferred. The choosing of k is thus a trade-off between smoothness of the estimate, and settling time of the estimate.

The median filter is another example of a filter used for noise removal, being suited for the removal of outliers (large sporadic errors) affecting measurements. Its expression is given by $v^*(n) = median(v(n-k), v(n-k+1), \ldots, v(n))$. Figure 3.9a shows the results of this filter for different values of k.

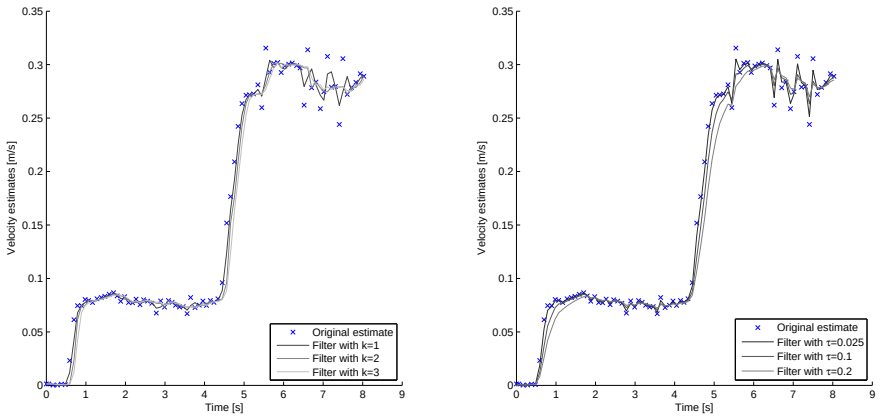
Once again, the larger the k, the smoother the response. One big advantage the median filter offers, as compared to the low pass-like filter, is its fast settling time. As observed in the figure, the estimate quickly converges to the original estimated value, while also performing some smoothing. Once again the trade-off in the choosing of k is between smoothness of the estimate, and settling time of the estimate.

A continuous low pass filter can be approximated by a discrete implementation, resulting in the exponentially-weighted auto-regressive filter.

The filter output is computed using $v^*(n) = \alpha v(n) + (1-\alpha)v^*(n-1)$, where $\alpha = \frac{t_{step}}{(t_{step}+\tau)}$, and τ is the filter time constant.

Some results for different values of τ are shown in figure 3.9b.

The results are somewhat similar to the low pass-like filter mentioned above. The larger the τ the smoother the answer, however, it also increases its settling time. A smaller τ results in a faster settling time, but the noise is less attenuated.



(a) Median filter results, with different k values

(b) Discretization of low pass filter, with different time constants τ

Figure 3.9

The median filter, with $k = 2$, will be used henceforth for estimation of linear velocities.

Angular Velocity

The angular velocity is computed as:

$$\omega(t) = \frac{\theta(t) - \theta(t - t_{step})}{t_{step}}$$

An experiment is setup as follows:

> *A fixed voltage is applied as the steering command, hopefully resulting in an also fixed steering angle. A piece-wise linear voltage is then applied as the velocity command. This will result in the truck performing circles with a varying linear speed. The angular speed will also vary in accordance with the linear speed.*

The applied voltage as steering command is the same a the one use for the linear velocity seen in 3.7.

Applying this voltage resulted in the angular speed estimation of figure 3.10. As in the linear velocity case, there seems to be some unwanted noise, therefore a filtering procedure should provide better results. Only the median filter results are shown, since they outperformed the other methods (low pass-like filter and discretization of low pass filter).

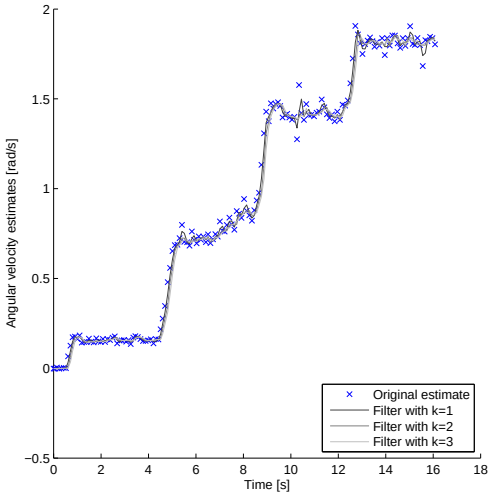


Figure 3.10: Median filter applied to the angular velocity estimates

A value of $k = 2$ is chosen. The remarks mentioned in the linear velocity case are applied here as well.

3.2.4 Steering Angle

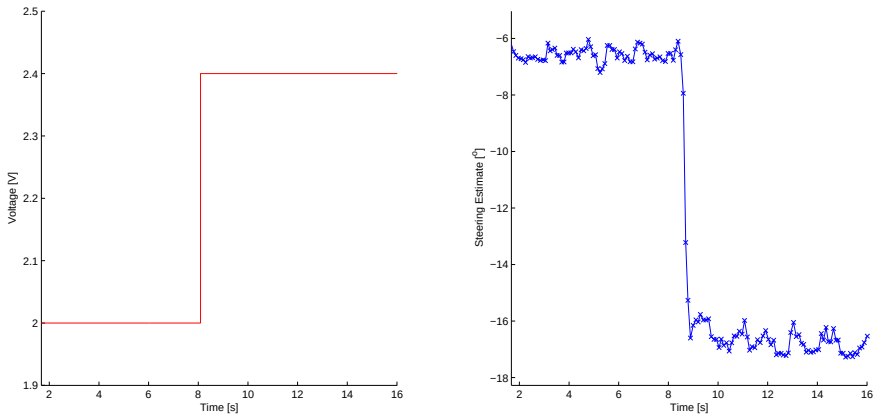
The following experiment was setup:

> *A fixed voltage is applied as the velocity command. A step function is applied as the steering command. This will result in the truck changing between different steering angles during the experiment.*

The steering voltage applied is seen in figure 3.11a.

The steering angle is estimated according to the car kinematic model as follows: $\phi = \arctan\left(\frac{L}{V}\omega\right)$. Where L is the distance between the front and rear axles of the truck. The values used for the linear and angular velocities are the already filtered ones, in order to improve the estimate of ϕ. The corresponding ϕ estimate is shown in figure 3.11b.

From the graph we can see that the steering angle estimates have a nice behaviour, probably as a result of the filtering process already done in the linear and angular velocity estimates. Therefore the steering angle estimate does not need any further filtering.



(a) The steering voltage applied

(b) Estimated steering angle

Figure 3.11

3.3 Actuator Delays

The actuator delay refers to the interval of time between an input command is issued until the time the corresponding output corresponding reaction settles on its final value.

3.3.1 Linear Velocity actuator

The following experiment setup was made:

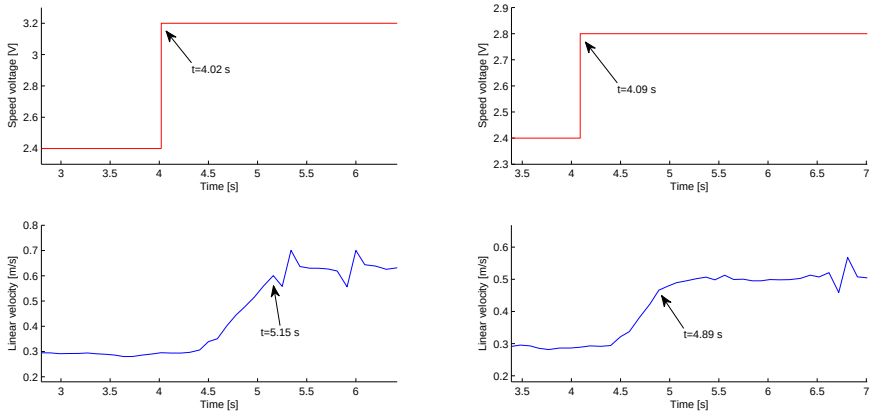
> *The truck steering angle is set to zero, so that it moves in a straight line. A step voltage is applied to the truck, the transition time happens during the experience. The time between the transition is issued until the actuator settles in the new velocity is then computed.*

Figures 3.12a, 3.12b and 3.13a are the result of the previously mentioned experiment for several set point combinations.

It is noticeable that the delay increases with the magnitude of the set point difference. The measured delays are between 0.5 s and 1.1 s.

3.3.2 Steering Angle actuator

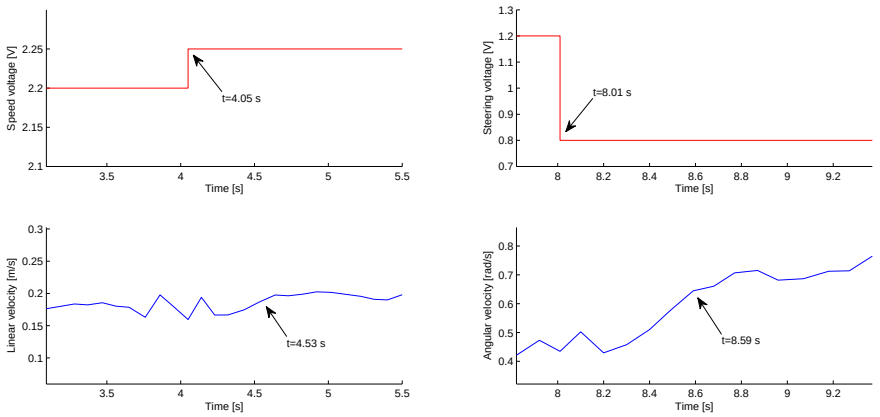
The experimental setup is hereby described:



(a) A measured delay/settling time of 1.13 s for an increase in speed of 0.3 m/s

(b) A measured delay/settling time of 0.80 s for an increase in speed of 0.2 m/s

Figure 3.12



(a) The delay is hard to measure for very small increases in speed

(b) Estimating the delay/settling time in the steering angle actuator

Figure 3.13

A constant voltage is applied as linear velocity command, thus resulting in a fixed truck speed. A step function voltage is applied as steering angle command, resulting in a change in the steering angle. The temporal distance between the voltage transition and the effective change in the measured angular velocity is measured, thus yielding the steering actuator delay.

The experiment results are shown in figure 3.13b.

At $t = 8.01s$ a new steering voltage is issued, and at $t = 8.59s$ the measured angular velocity seems to have settled around its final constant value. This results in an estimated delay of $\Delta t = 8.59 - 8.01 = 0.58s$. Several tests like this were performed, around different set points, in all of them the measured delay was in a neighbourhood of $0.6 \pm 0.1s$. It should be noted that the delay grows with the magnitude of the difference between set points.

3.4 Actuator-voltage relation

An important step in controller design is to have a rough estimate on the relation between the input and output. This offers a valuable *a priori* knowledge of the approximate input that the system should be fed, in order to output a desired value. This section is concerned with finding relation from input voltages to linear velocity and steering angle.

3.4.1 Linear Velocity

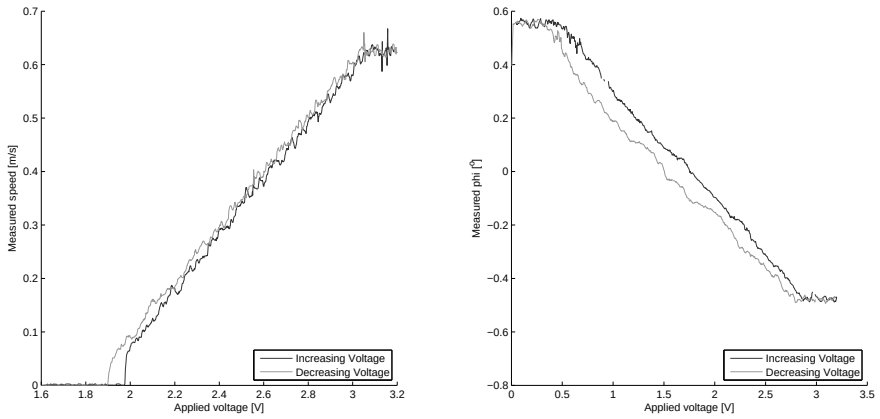
For the linear velocity case, the experimental setup is as follows:

The steering angle is fixed and different than zero, such that the truck moves in circles. A triangular voltage wave is applied as velocity command. This wave has a very large period such that the delay effects can be neglected.

Figure 3.14a is a result of this experiment:

We can observe that when the applied voltage is near $1.6V$ the actuator has no kind of measurable output. This effect is known as a deadband, or neutral zone, and it extends somewhere from $1.6V$ to $1.9V$ approximately. In the increasing voltage curve, this deadband seems to extend further than the $1.9V$ reaching almost $2.0V$, this is due to the inertia felt by the truck when it is stopped and trying to start moving.

Near high voltages, from $3.0V$ to $3.2V$ the measured speed stops changing, this can be identified as actuator saturation, the motor has reached its maximum velocity, and will not increase it any more, even though the voltage keeps rising.



(a) The voltage was slowly changed along the duration of the experience, around 240 seconds

(b) Estimating the relation between steering voltage and steering angle, the total experiment time was 120 seconds

Figure 3.14

3.4.2 Steering Angle

The experimental setup developed was:

> *A fixed voltage is applied as the linear velocity command. A triangle wave is applied as the voltage commanding the steering angle. This wave is as slowly changing as possible so that the inherent delay of the actuators can be neglected. The estimated steering angle is recorded as well as the applied steering voltage.*

Figure 3.14b results from the experience performed.

One can notice the gap between the Increasing and Decreasing Voltage curves, assuming that this is not an effect of the actuator delay, since the command voltage is slowly changing, it can be identified as an hysteresis phenomenon.

It is also observable that when near the limits of the applied voltage, the steering angle stops changing. This is associated with the actuator saturation, even though the voltage keeps increasing (or decreasing), the wheels already achieved its maximum reachable steering angle.

3.5 Making use of the Modelling

The modelling process described above provides an important insight into the system. It is also useful for controlling purposes both in simulation and in

practice.

3.5.1 Simulations

In order to save time and resources, control engineers often make use of simulation tools in order to test controller implementations before actually implementing them on the real system. We created our own simulator in MATLAB, which makes use of all the information retrieved before, like sensor noise and actuator non-linearities.

By doing so we can quickly and safely test controller implementations, before testing them on the trucks. The tuning of the controller parameters also becomes easier, and can give us a good starting point for good parameters in the real truck.

3.5.2 Practice

A great deal of challenges in the Control field is related to the sensing area, which is concerned with getting good measurements. In order to improve our control system, we make use of the previously tested filters, so that we can get good estimate of the measurements needed for control algorithms.

The voltage to linear and steering angle relations are also of great importance, for without their knowledge we would not be able to control the trucks in a reliable manner.

The Modelling is thus a process related to the Control problem itself, and as such, it becomes necessary to delve into it.

Chapter 4

Trajectory Tracking Controllers

In this chapter we will present some controllers for trajectory tracking.

The controllers presented were all designed for a single car without considering the existence of a trailer. For some of the controllers we present extensions which can take into account an attached trailer.

The trajectory tracking task is concerned about making a car follow a determined trajectory, which is defined as a path with an associated timing law. A more precise explanation is given in section 4.1.

After that, section 4.2 will present a simple controller, which has no feedback action. The objective of this section is to ease the reader into the following sections, and to explain the need for feedback.

The next two controllers to be implemented are well established in the current literature. They will be explained in sections 4.3 and 4.4. Another controller, based on Sliding Mode Control is introduced in section 4.5 alongside with a brief introduction to Sliding Mode Control. A last controller, based in the current trends in the already existing technologies in automotive industry is implemented in section 4.6.

4.1 Trajectory Tracking

When developing an autonomous mobile system, one has many degrees of freedom from which to choose. Concerning the navigation part of our system, one could have chosen either a Path Following Controller or a Trajectory Tracking Controller, the difference between both, is subtle but important [29], and will now be explained.

A path is a set of points, which codify sequential states, starting at an initial configuration, and ending at a goal configuration. In our case, a path, is simply given by a set points in the ground plane,$(x_i, y_i) \in \mathbf{R}^2$. Imagining we have a

set of N points defining a path, it can be expressed as

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_k, y_k), \ldots, (x_{N-1}, y_{N-1})\}$$

where (x_0, y_0) is the initial position of the path, and (x_{N-1}, y_{N-1}) is the final position, or goal position of the path.

A trajectory can be seen as a particular realization of a path, having the exact same sequence of points as the previous path example, but with an additional time coordinate $t_i \in \mathbf{R}_0^+$ for every point, resulting in an additional set of time coordinates for the path. A trajectory can be represented as

$$\{(x_0, y_0, t_0), (x_1, y_1, t_1), \ldots, (x_k, y_k, t_k), \ldots, (x_{N-1}, y_{N-1}, t_{N-1})\}$$

where similarly, (x_0, y_0, t_0) and $(x_{N-1}, y_{N-1}, t_{N-1})$ are the initial and final positions.

A Path Following Controller will focus on converging to the specified path, however it will do so in the easiest way available. The easiest way can consist in going to the nearest point in the path or the path point which is right in front of the car.

Figure 4.1 shows an illustration of a path following controller, which tries to converge to the nearest point on the path.

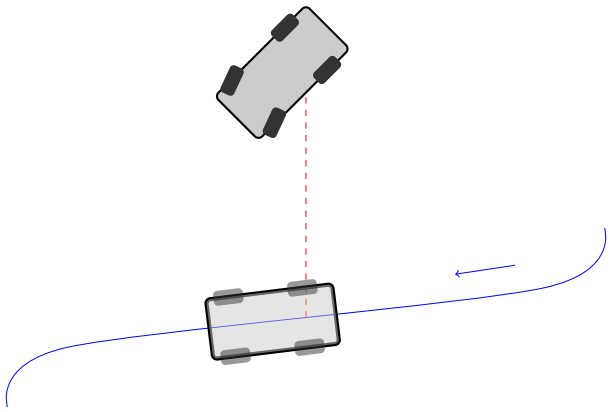


Figure 4.1: An illustration of the path following task

A Trajectory Tracking Controller will focus on converging to a specified point of the path, this point is chosen as the one with the same (or closest) time coordinate t_i as the current time t. Figure 4.2 shows the trajectory tracking objective.

One can say that the trajectory tracking problem is harder than the path following one, simply by noticing that the path following has the liberty of choosing a convenient path point for which to converge. In the trajectory tracking task (figure 4.2), the controller will impose input commands, such that it tries to take the truck from the current position to the desired position as given by the trajectory and the current time, the controller can then result in an aggressive

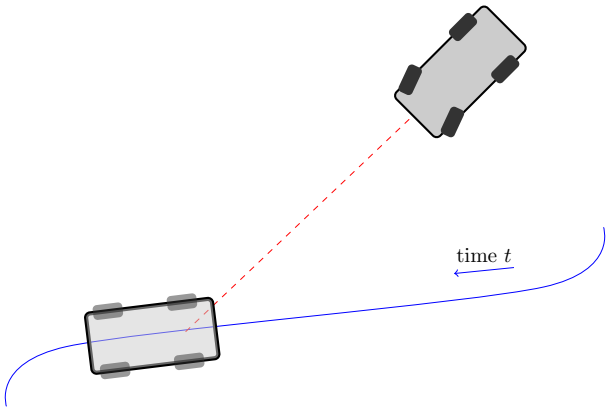


Figure 4.2: An illustration of the trajectory tracking task

behaviour, since it is trying to compensate a very large error. However if a path following controller were to be used, the desired position would instead be a convenient (*e.g.* the closest, as seen in figure 4.1) point in the path. This would result in a lower error, and consequently, in a smoother behaviour of the controller.

This reason would be enough to convince one to go with a path following controller instead of a trajectory tracking one. However using a trajectory tracking controller presents some benefits, like the ability to set a desired speed for specific sections of the trajectory. This is of the utmost importance, since in real situations, we would like our truck to travel at an adequate speed (not too slow and not too fast). Also, being that we are in a somewhat controlled environment, we can always assume that the truck is never dangerously far from the trajectory, since a trajectory will always start from the initial position of the truck (or in a close neighbourhood). The only way for the truck to deviate dangerously from the desired trajectory is if the controller allows so, however this would render the controller useless.

The problem of trajectory tracking in wheeled mobile robots is well studied in the literature. Most of the developed work is focused on controlling a single car or unicycle-like vehicle, thus ignoring the existance of a trailer. This is not a problem for our case, as we will explain later. Some of the possible directions of research that we did not dwelve into will now be surveyed. Model predictive controllers can be used for this task, as seen in [30],[31],[32] and [33]. There has been some approaches which seek to use adaptive controllers, in order to deal with system uncertainties, some examples are found in [34],[35],[36] and [37]. A comparison of several controllers can be found in [38].

Some research however has been made, which takes into account the existence of a trailer, as seen in [39] and [40]. Some alternative approaches to solve this problem include the limitation of steering [41] and a particle swarm optimization algorithm [42]. Some solutions are proposed for the backward movement of a truck and trailer system, which represents a highly unstable

system, these can be found in [43] and [44].

4.2 Feedforward Controller

In this section we will introduce the Feedforward Controller. Its name stems from the fact that all of the commands applied to the system to be controlled are computed *a priori*, and there is no kind of feedback action in order to compensate possible errors in the trajectory tracking task.

4.2.1 The Car Model

The kinematic car model will be used to model our truck. Remembering the equations:

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= \frac{v}{R}\tan\phi \end{aligned} \tag{4.1}$$

The input commands, which are set by the controller are v and ϕ, corresponding to the linear velocity and the steering angle. To simplify the design of the upcoming controllers, the change of variable (4.2) will be applied, resulting in the diffeomorphism given by equation (4.3).

$$\omega = \frac{v}{R}\tan\phi \tag{4.2}$$

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= \omega \end{aligned} \tag{4.3}$$

The reverse change of variable of (4.2), is given by:

$$\phi = \arctan\left(\frac{R}{v}\omega\right) \tag{4.4}$$

System (4.3) is known as the kinematic unicycle model. Our control inputs are now v and ω, where ω represents the angular velocity of the truck.

4.2.2 Obtaining the Feedforward Commands

The feedforward commands are defined as the set of commands that should be applied to system (4.3), such that it follows a specified trajectory. As we are going to see ahead, these commands can be computed before the actual trajectory is followed, hence resulting in the designation of feedforward controller.

Assuming we have a trajectory given by the following set of points:

$$\{(x_0, y_0, t_0), (x_1, y_1, t_1), \ldots, (x_k, y_k, t_k), \ldots, (x_{N-1}, y_{N-1}, t_{N-1})\} \tag{4.5}$$

We can solve system (4.3), in order to obtain the linear velocity input command v. By squaring and summing the first two equations of the system we get

$$v^2 = \dot{x}^2 + \dot{y}^2 \equiv v = \sqrt{\dot{x}^2 + \dot{y}^2} \tag{4.6}$$

We can approximate the derivative of x by the difference quotient:

$$\dot{x}_k = \frac{x_k - x_{k-1}}{t_k - t_{k-1}} \tag{4.7}$$

Computing the derivatives $\dot{x}_k$ and $\dot{y}_k$, associated to the trajectory (4.5), we can then get v_k from (4.6).

Similarly, one can compute the angular velocity command ω. To a path (x_r, y_r) corresponds a uniquely defined θ_r, which can be obtained by the non holonomic constraints of the car. Since θ corresponds to the direction of movement, one has

$$\theta = \arctan\left(\frac{\dot{y}}{\dot{x}}\right) \tag{4.8}$$

Differentiating w.r.t time (4.8) we get:

$$\omega = \dot{\theta} = \frac{\dot{x}\ddot{y} - \ddot{x}\dot{y}}{\dot{x}^2 + \dot{y}^2} \tag{4.9}$$

Which results in the approximation

$$\omega_k = \frac{\dot{x}_k\ddot{y}_k - \ddot{x}_k\dot{y}_k}{\dot{x}_k^2 + \dot{y}_k^2} \tag{4.10}$$

The input commands v_k and ω_k were computed in a straight forward manner using the trajectory and the car model. What has been done was simply compute the commands that have to be applied such that the specified trajectory is performed. If these exact commands are applied to the truck, and if its initial position is coincident with the trajectory initial position $(x_0, y_0, 0)$ the truck should then perform the specified trajectory precisely.

However this is not true, the reason being that the kinematic model expressed in (4.3) is a rough approximation of the truck behaviour, and does not explain exactly its time evolution.

Some of the unmodelled phenomena are the higher order dynamics, remember that accelerations are not modelled here, which results in assuming that the truck can have instantaneous changes of speed (corresponding to an impossible infinite acceleration). Another unmodelled dynamic is usually related to wheel friction. For example wheel drifting happens when fast changes of speed

occur (big accelerations or brakings) or when turning at high speeds (sideways movement due to wheel slippage).

We also assume a perfect knowledge of the parameters used in the kinematic model (in the truck case the axle distance L), which can be measured with high accuracy but are never known exactly. Another point worth mentioning relates to the difficulty in applying an exact command v_k and ω_k. For the linear velocity case, it is the result of a motor applying a torque on the wheels, forcing the motor to apply the correct torque such that it results in the desired linear speed is in itself a control problem. The same can be said for the angular velocity, or equivalently, steering angle, which also needs its own controller in order to output the desired reference angle.

These problems illustrate the need for a feedback action [45], that is, a control algorithm that makes use of the current error information. Simply put, the feedback control should perceive the current situation and act based on it.

One can heuristically develop a simple feedback controller, algorithm 5 shows a possible solution.

```
initialization;
while not at the end of trajectory do
    Get current position;
    Get current trajectory point;
    if behind trajectory point then
        increase velocity;
    else
        decrease velocity;
    end
    if to the left trajectory point then
        steer to the right;
    else
        steer to the left;
    end
end
```

Algorithm 5: An heuristic trajectory tracking controller

Along the following sections we will develop feedback controllers, however we will leave the heuristic approach and use established Control tools to create controllers which perform this task in a satisfactory and reliable way.

4.3 Linearization Based Controller

This control design procedure can be found in [46]. In order to better introduce this section, lets start by looking at a typical control loop, as seen in figure 4.3.

One of the first steps is to define an error function (corresponding to e in

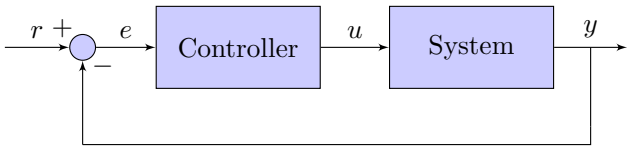


Figure 4.3: Control loop

figure 4.3) for the truck. The task of the controller will then consist in forcing this error to zero.

Considering that the truck state is given by (x, y, θ) and the desired reference state (corresponding to r in figure 4.3) is given by (x_r, y_r, θ_r), we can define the error function as illustrated on figure 4.4.

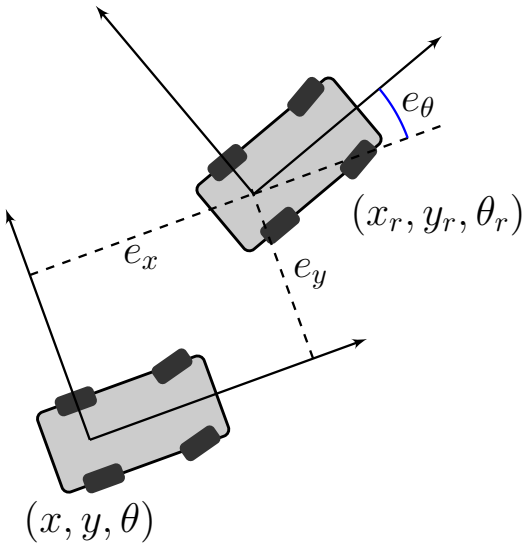


Figure 4.4: State error

Computed as

$$\begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r - x \\ y_r - y \\ \theta_r - \theta \end{bmatrix} \tag{4.11}$$

We can differentiate (4.11) with respect to time, and obtain the error dynamic equations. These equations define the evolution of the error in time. Differentiation yields:

$$\begin{aligned}\dot{e}_x &= \cos(e_\theta)v_r - v + e_y\omega \\ \dot{e}_y &= \sin(e_\theta)v_r - e_x\omega \\ \dot{e}_\theta &= \omega_r - \omega\end{aligned} \tag{4.12}$$

The subscript r (as in v_r or ω_r) refers to the reference feedforward commands, whereas v or ω are be the current linear and angular velocities.

We can apply input substitution:

$$\begin{aligned}u_1 &= -v + v_r \cos e_\theta \\ u_2 &= \omega_r - \omega\end{aligned} \tag{4.13}$$

to (4.12) in order to get the simplified error dynamics:

$$\begin{bmatrix}\dot{e}_x \\ \dot{e}_y \\ \dot{e}_\theta\end{bmatrix} = \begin{bmatrix}0 & \omega & 0 \\ -\omega & 0 & 0 \\ 0 & 0 & 0\end{bmatrix}\begin{bmatrix}e_x \\ e_y \\ e_\theta\end{bmatrix} + \begin{bmatrix}0 \\ \sin e_\theta \\ 0\end{bmatrix} v_r + \begin{bmatrix}1 & 0 \\ 0 & 0 \\ 0 & 1\end{bmatrix}\begin{bmatrix}u_1 \\ u_2\end{bmatrix} \tag{4.14}$$

The next step is to linearise the system around the equilibrium point $\mathbf{e} = \mathbf{0}$ and $\mathbf{u} = \mathbf{0}$, this is the same as linearising the system around the desired trajectory. The linearisation yields:

$$\begin{bmatrix}\dot{e}_x \\ \dot{e}_y \\ \dot{e}_\theta\end{bmatrix} = \begin{bmatrix}0 & \omega_r & 0 \\ -\omega_r & 0 & v_r \\ 0 & 0 & 0\end{bmatrix}\begin{bmatrix}e_x \\ e_y \\ e_\theta\end{bmatrix} + \begin{bmatrix}1 & 0 \\ 0 & 0 \\ 0 & 1\end{bmatrix}\begin{bmatrix}u_1 \\ u_2\end{bmatrix} \tag{4.15}$$

If we assume a trajectory with constant linear and angular velocities, v_r and ω_r, system (4.15) becomes time invariant. Computing its controllability matrix, we can infer if the states of the system can be altered through adequate input commands, *i.e.*, if we can devise a linear control law that makes the system stable. The controllability matrix for a system of dimension N is given by $C = [BAB \dots A^{N-1}B]$. If $rank(C) = N$, then the system is controllable, and a suitable linear control law can be applied such that the system is rendered stable. In our case, the controllability matrix is given by:

$$C = [BABA^2B] = \begin{bmatrix}1 & 0 & 0 & 0 & -\omega_r^2 & -v_r\omega_r \\ 0 & 0 & -\omega_r & -v_r & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0\end{bmatrix} \tag{4.16}$$

The rank of C is 3 (there are at least 3 linearly independent columns), except when v_r and ω_r are simultaneously zero. This means that our system is controllable for trajectories which are straight lines or arcs of circumference.

Unfortunately, we cannot use this type of analysis for more complex trajectories (consisting of multiple straight lines and arcs of circumference), since the linearised system (4.15) would become time invariant.

We can however try to intuitively reason that if we can control along single (invariant) trajectories, then we might be able to control the system along a trajectory consisting of these smaller invariant trajectories.

Getting back to our control design, we choose the input u to be

$$\begin{aligned} u_1 &= -k_1 e_x \\ u_2 &= -k_2 sign(v_r) e_y - k_3 e_\theta \end{aligned} \tag{4.17}$$

Resulting in the autonomous system

$$\dot{\mathbf{e}} = \begin{bmatrix} -k_1 & \omega_r & 0 \\ -\omega_r & 0 & v_r \\ 0 & -k_2 sgn(v_r) & -k_3 \end{bmatrix} \mathbf{e} \tag{4.18}$$

We can compute the characteristic equation of system (4.18) (of the type $\dot{x} = Ax$) as $|\lambda I - A| = 0$, where I is the $n \times n$ identity matrix, and $|.|$ stands for the matrix determinant, and choose controller gains k_1, k_2 and k_3 such that it matches the more familiar characteristic equation:

$$(\lambda + 2\xi a)(\lambda^2 + 2\xi a\lambda + a^2) \tag{4.19}$$

Doing so we get

$$\begin{aligned} k_1 &= k_3 = 2\xi a \\ k_2 &= \frac{a^2 - w_r^2}{|v_r|} \end{aligned} \tag{4.20}$$

By choosing ξ and a to be positive values, we can ensure stability (only for an invariant trajectory), since the characteristic equation (4.19) will have all of its poles in the left half complex plane. Also, parameters ξ and a can be tuned in order to obtain desired response characteristics (as small settling time or low overshoot).

One can see that when the reference speed v_r approaches zero, the gain k_2 tends to infinity, possibly resulting in a infinitely large input command u_2. A possible solution to this problem is to redefine the control gains as:

$$\begin{aligned} k_1 &= 2\xi\sqrt{w_r^2 + bv_r^2} \\ k_2 &= b|v_r| \\ k_3 &= 2\xi\sqrt{w_r^2 + bv_r^2} \end{aligned} \tag{4.21}$$

This however will make the roots of the characteristic equation vary with different values of v_r and ω_r. The stability of this system cannot be proven, since it is time-varying, however we can use the previous argument, that a trajectory consists of several smaller invariant trajectories to convince ourselves that the controller will be stable in practice.

When $v_r = \omega_r = 0$, gains k_1,k_2 and k_3 are simply zero, resulting in no action whatsoever from the truck (motion stopped). This is a simple way to solve the controllability problem around trajectories with v_r and ω_r equal to zero. By simply stopping the truck, we ensure that the error will not grow unbounded.

Once we have our input u_1 and u_2 defined, we know the desired linear and angular speed that the truck should have. It is simply given by reverting input tranformation (4.13), thus obtaining

$$\begin{aligned} v &= -u_1 + v_r \cos e_\theta \\ \omega &= \omega_r - u_2 \end{aligned} \tag{4.22}$$

4.3.1 On the forward movement stability of the Trailer

In this section we will do a brief study on the stability of the trailer when the truck is moving forward. The objective is to show that the use of controllers that ignore the trailer is suitable for the tracking task, since the trailer will be a stable system. We start by using an alternative kinematic model for the truck and trailer system, given by

$$\begin{aligned} \dot{x} &= v \cos\theta_0 \\ \dot{y} &= v \sin\theta_0 \\ \dot{\theta_0} &= \omega \\ \dot{\theta_1} &= \frac{v}{L} \sin\left(\theta_0 - \theta_1\right) \end{aligned} \tag{4.23}$$

Where (x, y, θ_0) represents the pose of the truck pulling the trailer (equivalent to the car kinematic model (4.3)). And θ_1 represents the orientation of the trailer. Figure 4.5 shows an illustration of the truck and trailer and the state variables associated to it. The addition of a trailer only increases the dimensionality of the state by one, since if the orientation of the trailer and the position of the truck rear axle are known, the trailer position is computed in a straight forward manner.

Similarly to the car case, equation (4.11), the state error for the truck trailer system can be defined as

$$\begin{bmatrix} e_x \\ e_y \\ e_{\theta_0} \\ e_{\theta_1} \end{bmatrix} = \begin{bmatrix} \cos\theta_0 & \sin\theta_0 & 0 & 0 \\ -\sin\theta_0 & \cos\theta_0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r - x \\ y_r - y \\ \theta_{r_0} - \theta_0 \\ \theta_{r_1} - \theta_1 \end{bmatrix} \tag{4.24}$$

The first three rows of (4.24) are the original car state error (4.11), while the fourth row is an additional equation, representing the trailer orientation error.

The following step consists in computing the error dynamics. The first equations (regarding $\dot{e}_x$,$\dot{e}_y$ and $\dot{e}_{\theta_0}$) remain the same as (4.12). The dynamic of $\dot{e}_{\theta_1}$ is computed as follows:

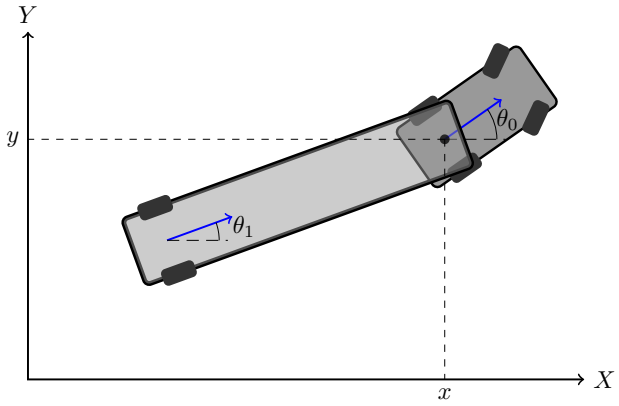


Figure 4.5: Illustration of the Truck and Trailer system and its corresponding state variables

$$
\begin{aligned}
e_{\theta_1} &= \theta_{r_1} - \theta_1 \\
\dot{e}_{\theta_1} &= \dot{\theta}_{r_1} - \dot{\theta}_1 \\
&= \frac{v_r}{L}\sin\left(\theta_{r_0} - \theta_{r_1}\right) - \frac{v}{L}\sin\left(\theta_0 - \theta_1\right) \\
&= \frac{v_r}{L}\sin\left(e_{\theta_0} - e_{\theta_1} + \theta_0 - \theta_1\right) - \frac{v}{L}\sin\left(-e_{\theta_0} + e_{\theta_1} + \theta_{r_0} - \theta_{r_1}\right)
\end{aligned} \tag{4.25}
$$

Applying again input substitution (4.13), the full error dynamic equations are given by:

$$
\begin{aligned}
\dot{e}_x &= \cos e_3 v_r - v + e_2\omega \\
\dot{e}_y &= \sin e_3 v_r - e_1\omega \\
\dot{e}_{\theta_0} &= \omega_r - \omega \\
\dot{e}_{\theta_1} &= \frac{v_r}{L}\left(\sin\left(e_{\theta_0} - e_{\theta_1} + \theta_0 - \theta_1\right) - \cos\left(e_{\theta_0}\right)\sin\left(-e_{\theta_0} + e_{\theta_1} + \theta_{r_0} - \theta_{r_1}\right)\right) \\
&\quad + \frac{u_1}{L}\sin\left(e_{\theta_0} - e_{\theta_1} + \theta_{r_0} - \theta_{r_1}\right)
\end{aligned} \tag{4.26}
$$

The following step consists in linearising system (4.26) around the trajectory (corresponding to the equilibrium point $\mathbf{e} = \mathbf{0}$,$\mathbf{u} = \mathbf{0}$). Only the linearised equation of $\dot{e}_{\theta_1}$, needs to be computed, since the other linearised state variable dynamics remain the same as (4.15). The linearised dynamics can be computed as:

$$
\dot{e}_{\theta_1} = \frac{\partial \dot{e}_{\theta_1}}{\partial e_x}e_x + \frac{\partial \dot{e}_{\theta_1}}{\partial e_y}e_y + \frac{\partial \dot{e}_{\theta_1}}{\partial e_{\theta_0}}e_{\theta_0} + \frac{\partial \dot{e}_{\theta_1}}{\partial e_{\theta_1}}e_{\theta_1} + \frac{\partial \dot{e}_{\theta_1}}{\partial u_1}u_1 + \frac{\partial \dot{e}_{\theta_1}}{\partial u_2}u_2|_{\mathbf{e}=\mathbf{0},\mathbf{u}=\mathbf{0}} \tag{4.27}
$$

The first, second and last terms of the sum in (4.27) are equal to zero, since $\dot{e}_{\theta_1}$ does not depend on e_x, e_y or u_2. The remaining terms are found to be:

$$\begin{aligned}
\frac{\partial \dot{e}_{\theta_1}}{\partial e_{\theta_0}}|_{\mathbf{e=0,u=0}} &= 2\frac{v_r}{L}\cos\left(\theta_{r_0} - \theta_{r_1}\right) \\
\frac{\partial \dot{e}_{\theta_1}}{\partial e_{\theta_1}}|_{\mathbf{e=0,u=0}} &= -2\frac{v_r}{L}\cos\left(\theta_{r_0} - \theta_{r_1}\right) \\
\frac{\partial \dot{e}_{\theta_1}}{\partial e_{u_1}}|_{\mathbf{e=0,u=0}} &= \frac{1}{L}\sin\left(\theta_{r_0} - \theta_{r_1}\right)
\end{aligned} \tag{4.28}$$

Resulting in the full linearised dynamics

$$\begin{aligned}
\begin{bmatrix} \dot{e}_x \\ \dot{e}_y \\ \dot{e}_{\theta_0} \\ \dot{e}_{\theta_1} \end{bmatrix} &= \begin{bmatrix} 0 & \omega_r & 0 & 0 \\ -\omega_r & 0 & v_r & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2\frac{v_r}{L}cos\left(\theta_{r_0} - \theta_{r_1}\right) & -2\frac{v_r}{L}cos\left(\theta_{r_0} - \theta_{r_1}\right) \end{bmatrix} \begin{bmatrix} e_x \\ e_y \\ e_{\theta_0} \\ e_{\theta_1} \end{bmatrix} \\
&+ \begin{bmatrix} & 1 & 0 \\ & 0 & 0 \\ & 0 & 1 \\ \frac{1}{L}\sin\left(\theta_{r_0} - \theta_{r_1}\right) & & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}
\end{aligned} \tag{4.29}$$

An intuitive analysis can be made, on the stability of the state variable e_{θ_1} corresponding to the trailer orientation. Assuming that $(e_x, e_y, e_{\theta_0}) = (0, 0, 0)$ and $u_2 = 0$, that is, the truck is already in the correct pose, the fourth row of (4.29) becomes

$$\dot{e}_{\theta_1} = -2\frac{v_r}{L}cos\left(\theta_{r_0} - \theta_{r_1}\right)e_{\theta_1} \tag{4.30}$$

Which is equivalent to the autonomous linear system $\dot{x} = Ax$, with $A = -2\frac{v_r}{L}cos\left(\theta_{r_0} - \theta_{r_1}\right)$. Considering that $|\theta_{r_0} - \theta_{r_1}| < \pi/2$ (that is the trailer and the truck are not in a Jackknife situation), we get that $cos\left(\theta_{r_0} - \theta_{r_1}\right) > 0$. We can make $\gamma = \frac{2}{L}cos\left(\theta_{r_0} - \theta_{r_1}\right)$, and knowing that $\gamma > 0$, we rewrite (4.30) as

$$\dot{e}_{\theta_1} = -\gamma v_r e_{\theta_1} \quad \gamma > 0 \tag{4.31}$$

Equation (4.31) is a linear differential equation and its solution is given by

$$e_{\theta_1}(t) = e_{\theta_1}(0)\exp(-\gamma v_r t) \tag{4.32}$$

If $-\gamma v_r < 0$, $\lim_{t\to+\infty} e_{\theta_1} = 0$, which means that the trailer will converge to the correct orientation. Since $\gamma > 0$, the condition for this to happen is $v_r > 0$.

Theorem 1. *For the truck and trailer system* (4.23) *linearised around the trajectory, the trailer orientation will converge to its desired value, if the truck is in the desired orientation, the trailer is not in a Jackknife situation and the truck is moving forward.*

Proof. Assume that

- The truck is in the desired orientation, $e_{\theta_0} = 0$;
- The truck and trailer are not in a Jackknife situation, *i.e.* $|\theta_{r_0} - \theta_{r_1}| < \pi/2 \equiv cos\,(\theta_{r_0} - \theta_{r_1}) > 0$
- The truck is supposed to move forward, $v_r > 0$.

Then, $\dot{e}_{\theta_1} = -2\frac{v_r}{L} cos\,(\theta_{r_0} - \theta_{r_1})\, e_{\theta_1}$, where $-2\frac{v_r}{L} cos\,(\theta_{r_0} - \theta_{r_1}) < 0$.
That is, $\dot{e}_{\theta_1} = f(\theta_{r_0}, \theta_{r_1}, v_r) e_{\theta_1}$ where $f(\theta_{r_0}, \theta_{r_1}, v_r) < 0$, thus resulting in a system that converges to the origin, $\lim_{t\to+\infty} e_{\theta_1} = 0$. □

The previous theorem states that the linearised trailer orientation error dynamics are stable when the truck and trailer are moving forward in a non extreme configuration (*i.e.* without Jackknife). Being so, one can simply ignore the trailer orientation error, and focus only on controlling the truck, since the trailer orientation will converge to the desired orientation.

4.3.2 Additional Error Dynamics

The previously presented controller, and all of its derivation is based on a state space approach. The typical and most common approach to control problems is usually done through the well known PID controller. In order to improve the performance we will mix the previously developed controller, with the well established properties of the PID control. The advantages of the PID controller come from its ability to eliminate steady state error, through its integral part, and to avoid overshoot, through its derivative component.

In order to combine both controllers, we simply create an alternative error, $\bar{e}$, which is the result of filtering the original error e, through a PID controller, resulting in

$$\bar{e}(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \dot{e}(t) \tag{4.33}$$

We apply equation (4.33) to all errors, e_x, e_y and e_θ. By preprocessing the error signals in this way, we are effectively adding the properties of a PID controller to our original controller. The downside is that we now have nine new parameters to tune, the proportional, integral and derivative gains of the errors corresponding to x,y and θ. A schematic representation can be seen in figure 4.6.

4.4 Nonlinear Controller

The Nonlinear Controller is based on a controller design method which proves the stability and convergence to the desired state using Lyapunov's Stability Theorem [47]. This controller is designed in order to control the system given in (4.14).

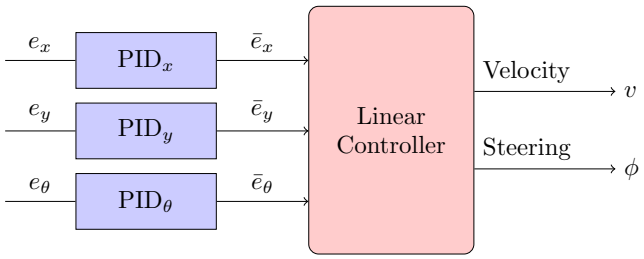


Figure 4.6: Schematic representation of the error dynamics extension

We will show that following control inputs given by

$$\begin{aligned} u_1 &= -k_1\left(v_r(t), w_r(t)\right) e_x \\ u_2 &= -\bar{k}_2 v_r(t) \frac{\sin e_\theta}{e_\theta} e_y - k_3\left(v_r(t), w_r(t)\right) e_\theta \end{aligned} \tag{4.34}$$

stabilize system (4.14). We define $\bar{k}_2$ as a constant such that $\bar{k}_2 > 0$, and $k_1\left(v_r(t), w_r(t)\right)$ and $k_3\left(v_r(t), w_r(t)\right)$ are defined as positive and continuous functions. These properties will play an important role in the following steps.

From now on k_1 and k_3 will be used to represent these functions, the dependence on $v_r(t), w_r(t)$ will be dropped for the sake of simplicity.

Substituting u_1 and u_2 as defined by (4.34) in (4.14), we get

$$\begin{aligned} \dot{e}_x &= \omega_r e_y - k_1 e_x \\ \dot{e}_y &= -\omega_r e_x + \sin(e_\theta) v_r \\ \dot{e}_\theta &= -\bar{k}_2 v_r \frac{\sin e_\theta}{e_\theta} e_y - k_3 \end{aligned} \tag{4.35}$$

Theorem 2. *If v_r and w_r are continuous and bounded and $v_r \not\to 0$ or $w_r \not\to 0$ for $t \to \infty$, then the controller given by* (4.34) *makes $\mathbf{e} = \mathbf{0}$ a globally asymptotically stable equilibrium of system* (4.14)*.*

Proof. Define the following Lyapunov function

$$V(t) = \frac{\bar{k}_2}{2}\left(e_x^2 + e_y^2\right) + \frac{e_\theta^2}{2}$$

Since we defined gains $\bar{k}_2$ (constant), k_1 and k_3 to be positive, we get that

$$V(t) \geq 0 \tag{4.36}$$

The time derivative along the solutions of system (4.35) is given by

$$
\begin{aligned}
\dot{V} &= \bar{k}_2 \left(e_x \dot{e}_x + e_y \dot{e}_y\right) + e_\theta \dot{e}_\theta \\
&= \bar{k}_2 \left(e_x \left(w_r e_y + u_1\right) + e_y \left(-w_r e_x + \sin(e_\theta) v_r\right)\right) + e_\theta u_2 \\
&= \bar{k}_2 \left(e_x u_1 + \sin e_\theta v_r e_y\right) + e_\theta u_2 \\
&= \bar{k}_2 \left(e_x(-k_x e_x) + \sin e_\theta v_r e_y\right) + e_\theta \left(-\bar{k}_2 v_r(t) \frac{\sin e_\theta}{e_\theta} e_y - k_3 e_\theta\right) \\
&= -k_1 \bar{k}_2 e_x^2 - k_3 e_\theta^2
\end{aligned} \tag{4.37}
$$

Once again, recalling that the gains are positive we have

$$
\dot{V} \leq 0 \tag{4.38}
$$

The following step in a regular Lyapunov analysis, would be to make use of invariant-set theorems to find the possible invariant set. The fact that the system is time-varying invalidates this approach, the alternative is to use Barbalat's theorem [48] which can be applied to our case in the following way.

If the following conditions are met:

- $V(e,t)$ is lower bounded; (verified by equation (4.36))
- $\dot{V}(e,t)$ is negative semi-definite; (verified by equation (4.38))
- $\dot{V}(e,t)$ is uniformly continuous in time (true due to the initial assumption of the gains k_1 and k_3 being continuous)

Then:

$$
\dot{V}(e,t) \to 0 \text{ as } t \to \infty \tag{4.39}
$$

Or the equivalent

$$
-k_1 \bar{k}_2 e_x^2 - k_3 e_\theta^2 \to 0 \text{ as } t \to \infty \tag{4.40}
$$

Following the original assumption that $v_r \not\to 0$ or $w_r \not\to 0$, then $k_1 \not\to 0$ and $k_3 \not\to 0$, which results in $e_x \to 0$ and $e_\theta \to 0$ when $t \to \infty$.

Studying system (4.35) for $t \to \infty$, and using the conclusions above we have

$$
\begin{aligned}
\dot{e}_x &= \omega_r e_y \\
\dot{e}_y &= 0 \\
\dot{e}_\theta &= -\bar{k}_2 v_r e_y
\end{aligned} \tag{4.41}
$$

Since $e_x \to 0$, then we also expect that $\dot{e}_x \to 0$, thus resulting in $e_y \to 0$.

Thus we have:

$$
(e_x, e_y, e_\theta) \to (0,0,0) \text{ as } t \to \infty \tag{4.42}
$$

□

The gains chosen are similar to the ones used for the Linear Controller in (4.21), and are defined as

$$\begin{aligned} k_1(v_r(t), \omega_r(t)) &= 2\xi\sqrt{w_r(t)^2 + bv_r(t)^2} \\ \bar{k}_2 &= b \\ k_3(v_r(t), \omega_r(t)) &= 2\xi\sqrt{w_r(t)^2 + bv_r(t)^2} \end{aligned} \tag{4.43}$$

Where $\xi > 0$ and $b > 0$, in order to abide by the assumptions needed to prove the stability of the controller.

The Nonlinear Controller developed is very similar to the Linear Controller. The main contribution of this section is to provide a Lyapunov-based analysis of the controller, which resulted in a proof of the global stability of the controlled linearised system, even for trajectories with a time variant v_r and ω_r.

4.5 Sliding Mode Controller

Sliding Mode Control (SMC) has been used extensively in modern control systems, its main advantage consists in high robustness to parametric uncertainties, unmodelled dynamics and exogenous disturbances.

The robustness to unmodelled dynamics is certainly an attractive property for the problem at hand. We are modelling our truck system using a simple kinematic model, so we expect the existence of unmodelled dynamics (as discussed in section 4.2), having a controller which is able to overcome these unknown factors is certainly desired.

The main drawback of this type of control is the presence of a very high frequency, low amplitude control command, which appears when the controlled system achieves its steady state. This so-called chattering effect is an immediate consequence of the control design.

The SMC comes to a trade-off decision, in which the designer must choose between performance or smooth controller commands. For a brief survey on SMC, the reader should refer to [49], or to [50] for a comprehensive explanation of SMC.

For this and the following section we will return to the car-like kinematic model given by (4.1). The design of this SMC is based on [51] and [52]. It is also convenient to define a new tracking error, which will simplify controller design and implementation. The new tracking error is defined as

$$\begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} = \begin{bmatrix} \cos\theta_r & \sin\theta_r & 0 \\ -\sin\theta_r & \cos\theta_r & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - x_r \\ y - y_r \\ \theta - \theta_r \end{bmatrix} \tag{4.44}$$

which simply corresponds to the error in the reference vehicle frame, instead of the error in the current vehicle frame.

SMC design consists in the definition of a suitable sliding surface s, and a controller that forces the system into that sliding surface. A sliding surface is

chosen to be a desired manifold of the state space, in which the system behaves in a desirable way.

Following [51], we define the desired sliding surfaces as

$$s_1 = \dot{e}_x + k_1 e_x \tag{4.45}$$

$$s_2 = \dot{e}_y + k_2 e_y + k_0 \operatorname{sgn}(e_y) e_\theta \tag{4.46}$$

The controller should force the system to the sliding surfaces, so that we have $s_1 = 0$ and $s_2 = 0$. Being on surface s_1 corresponds to equalling equation (4.45) to zero, which will make the errors, $\dot{e}_x$ and e_x tend to zero (if $k_1 > 0$). After convergence this corresponds to having a null longitudinal error, $e_x = 0$, and keeping it zero, $\dot{e}_x = 0$. It becomes hard to do the same analysis for surface s_2, however, intuitively, we are expecting that when on this surface (equation (4.46) equals to zero), we have $\dot{e}_y = 0$,$e_y = 0$ and $e_\theta = 0$, which correspond to a steady null lateral error e_y and a zero orientation error e_θ.

Once the sliding surfaces are defined, a reaching law must be derived. A reaching law determines the rate of the convergence to the the sliding surface.

The simplest, and most commonly used reaching law is given by

$$\dot{s} = -p \operatorname{sgn}(s) \tag{4.47}$$

We have for each of the surfaces $i = 1, 2$ that $|\dot{s}_i| = p_i$, thus resulting in reaching surface s_i at a constant rate given by p_i, more precisely reaching it in a finite time $t_i = \frac{s_i(0)}{p_i}$.

The reaching law used is a more complex version, which adds a proportional reaching rate. It can be written as

$$\dot{s} = -qs - p \operatorname{sgn}(s) \tag{4.48}$$

The proportional term $q > 0$ forces the state to approach the desired surface faster when the distance to the surface is greater. The reaching time is still finite and shorter than the one for reaching law give by equation (4.47).

For our system we get

$$\dot{s}_1 = -q_1 s_1 - p_1 \operatorname{sgn}(s_1) \tag{4.49}$$

$$\dot{s}_2 = -q_2 s_2 - p_2 \operatorname{sgn}(s_2) \tag{4.50}$$

Differentiating equations (4.45) and (4.46) w.r.t. time and replacing them in the left side of equations (4.49) and (4.50) results in

$$\ddot{e}_x + k_1 \dot{e}_x = -q_1 s_1 - p_1 \operatorname{sgn}(s_1) \tag{4.51}$$

$$\ddot{e}_y + k_2 \dot{e}_y + k_0 \operatorname{sgn}(e_y) \dot{e}_\theta = -q_2 s_2 - p_2 \operatorname{sgn}(s_2) \tag{4.52}$$

The first time derivatives of the error are computed to be

$$\begin{aligned}
\dot{e}_x &= -v_r - v\cos(e_\theta) + \frac{v_r}{L}\tan(\phi_r)e_y \\
\dot{e}_y &= v\sin(e_\theta) - \frac{v_r}{L}\tan(\phi_r)e_x \\
\dot{e}_\theta &= \frac{1}{L}\left(v\tan(\phi) - v_r\tan(\phi_r)\right)
\end{aligned} \tag{4.53}$$

The second derivative of the lateral error is given by

$$\ddot{e}_y = \dot{v}\sin(e_\theta) + v\cos(e_\theta)\dot{e}_\theta - \frac{1}{L}\left(v_r\tan(\phi_r) - \dot{v}_r\tan(\phi_r)e_x - v_r\frac{\dot{\phi}}{\cos^2(\phi_r)}e_x\right) \tag{4.54}$$

The second step of the SMC design consists in the design of the controller that forces the system to the desired sliding surfaces. In order to accomplish that we simply manipulate the above equations (4.51), (4.52), (4.53) and (4.54) and solve them for the inputs v and ϕ. Doing that results in

$$\dot{v} = \frac{1}{\cos e_\theta}\left(-q_1 s_1 - p_1\,\mathrm{sgn}(s_1) - k_1\dot{e}_x - \dot{\omega}_r e_y - \omega_r\dot{e}_y + v\dot{e}_\theta\sin(e_\theta) + \dot{v}_r\right) \tag{4.55}$$

$$\begin{aligned}
\phi = \arctan\Bigg(&\frac{L}{v}\omega_r + \frac{L}{v\left(v\cos(e_\theta) + k_0\,\mathrm{sgn}(e_y)\right)}\times \\
&\times\left(-q_2 s_2 - p_2\,\mathrm{sgn}(s_2) - k_2\dot{e}_y - \dot{v}\sin(e_\theta) + \dot{\omega}_r e_x + \omega_r\dot{e}_x\right)\Bigg)
\end{aligned} \tag{4.56}$$

Equations (4.55) and (4.56) give us the commanding inputs that should be applied to the truck system. Notice that we solved the first equation in order to $\dot{v}$ instead of v. We can numerically integrate $\dot{v}$ using the previous linear velocity command v, in order to find the linear velocity v^* to be applied at the current time instant.

We now must make the system converge to the desired sliding surfaces by choosing the right values for q_i and p_i.

Theorem 3. *If $q_i > 0$ and $p_i > 0$ for $i = 1, 2$ then the system will converge to the sliding surfaces s_1 and s_2.*

Proof. Defining the following Lyapunov function

$$V(s_1, s_2) = \frac{1}{2}(s_1^2 + s_2^2)$$

With $V(0) = 0$, $V(s_1, s_2) > 0\ \forall (s_1, s_2) \neq (0, 0)$ and $V(s_1, s_2) \to \infty$ as $\|(s_1, s_2)\| \to \infty$.

In order for (s_1, s_2) to be a global asymptotically stable equilibrium, the following must hold:

$$\dot{V}(s_1, s_2) < 0 \quad \forall\ (s_1, s_2) \neq (0, 0) \tag{4.57}$$

The Lyapunov time derivative can be written as

$$\dot{V} = s_1\dot{s}_1 + s_2\dot{s}_2$$

Replacing the surface time derivatives $\dot{s}_1$ and $\dot{s}_2$ by equations (4.49) and (4.50), we get

$$\begin{aligned}\dot{V} &= s_1\left(-q_1 s_1 - p_1\,\mathrm{sgn}(s_1)\right) + s_2\left(-q_2 s_2 - p_2\,\mathrm{sgn}(s_2)\right)\\ &= -q_1 s_1^2 - p_1|s_1| - q_2 s_2^2 - p_2|s_2|\end{aligned} \tag{4.58}$$

Equation (4.57) will be true if $q_i > 0$ and $p_i > 0$ for $i = 1, 2$.

□

The above theorem defines a suitable range of values for q_i and p_i, which guarantee global stability about the sliding surfaces s_i and finishes the design of the SMC.

4.5.1 Extension to Truck Trailer Case

To consider the Trailer we first need to extend the tracking error from equation (4.44) in order to account for the trailer orientation error. The extended tracking error is simply

$$\begin{bmatrix} e_x \\ e_y \\ e_{\theta_0} \\ e_{\theta_1} \end{bmatrix} = \begin{bmatrix} \cos\theta_r & \sin\theta_r & 0 & 0 \\ -\sin\theta_r & \cos\theta_r & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - x_r \\ y - y_r \\ \theta_0 - \theta_{r_0} \\ \theta_1 - \theta_{r_1} \end{bmatrix} \tag{4.59}$$

The design of a new controller consists in defining new suitable sliding surfaces. The dynamic equation of the trailer error given in (4.25) is repeated here for convenience

$$\dot{e}_{\theta_1} = \frac{v_r}{L}\sin\left(\theta_{r_0} - \theta_{r_1}\right) - \frac{v}{L}\sin\left(\theta_0 - \theta_1\right)$$

We notice that it is affected directly by the linear velocity input command v. As such we decided to couple the trailer orientation error e_{θ_1} with the longitudinal error e_x in the same sliding surface s_1 defined by

$$s_1 = \dot{e}_x + k_1 e_x + k_3\,\mathrm{sgn}(e_{\theta_1})e_{\theta_1} \tag{4.60}$$

Sliding surface s_2 is equivalent to the one defined in equation (4.46), and is

$$s_2 = \dot{e}_y + k_2 e_y + k_0\,\mathrm{sgn}(e_y)e_{\theta_0} \tag{4.61}$$

In order to compute the input commands, we differentiate w.r.t time the sliding surfaces (4.60) and (4.61) and manipulate the expressions until the we arrive at the control inputs, resulting in

$$\dot{v} = \frac{1}{\cos e_\theta}\left(-q_1 s_1 - p_1 \operatorname{sgn}(s_1) - k_1 \dot{e}_x - \dot{\omega}_r e_y - \omega_r \dot{e}_y \right.$$
$$\left. + v\dot{e}_{\theta_0} \sin(e_\theta) + \dot{v}_r + k_3 \operatorname{sgn}(e_{\theta_1})\dot{e}_{\theta_1}\right) \tag{4.62}$$

The steering angle input command ϕ is the same as the one defined in equation (4.56), however replacing e_θ by e_{θ_0}.

It was already proved that the system will converge to the sliding surfaces s_1 and s_2 if gains q_i and p_i are conveniently chosen to be positive.

This finishes the design of the SMC for the truck and trailer system.

4.6 Decoupled PID Controller

Nowadays several automotive manufacturers are researching and developing automatic systems with the objective of making the driving task safer. Some of these systems are already present in most of today's car (take for example ABS). In the near future we can expect to have more sophisticated systems implemented in our cars.

The objective of this section is then to try to develop a controller which can be based on the already available automotive technologies. This controller will be based on ideas presented in [53].

The control architecture will be such that the velocity and steering will be independently controlled by two decoupled controllers. The controller in charge of velocity commands will be called Longitudinal Controller, while the one who controls the steering wheels will be referred to as Lateral Controller.

One important point to mention is that this section will assume that the truck is moving on a structured road, for which the automotive technologies are developed and used on.

4.6.1 Lateral Controller

A current direction of research is Automated Lane Keeping. Driver tiredness and distraction can result in dangerous accidents. Recently developed technologies try to avoid this problem by alerting the driver when the car is leaving the lane.

These systems can be based on vision cameras. We will assume that these cameras can give us an estimate of the lateral error, *i.e.*, the distance of the car to the center of the lane. This error will be referred to as e_1. Besides this measurement assume also that the vision system is able to measure a look ahead error e_2. These errors are defined in figure 4.7.

The look ahead error e_2 is the lateral error that the truck will experience if it moves forward d_s meters while maintaining the current heading. This look ahead error can be seen as a prediction of the future lateral error, thus allowing to react faster to curves on the road.

The objective of the controller is to force both errors e_1 and e_2 to be zero. When $e_1 = 0$ we have that the truck is centered in the lane, and if additionally

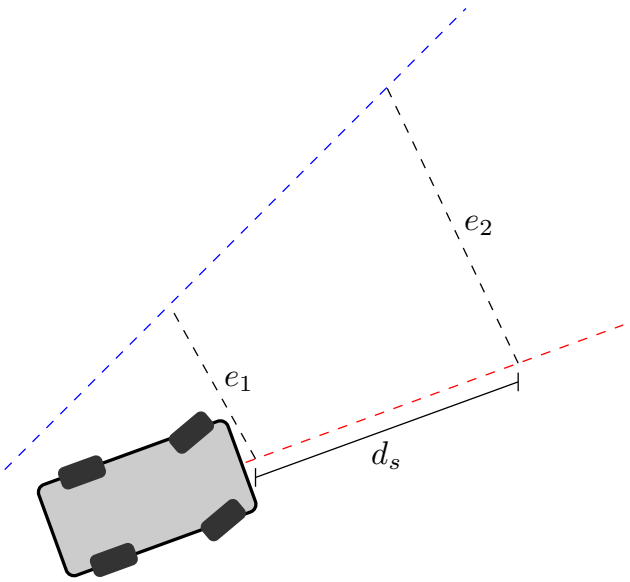


Figure 4.7: Lateral errors, current (e_1) and look ahead (e_2), measured at the look ahead distance, d_s metres in front of the vehicle

we have $e_2 = 0$ then the truck will also have a correct heading, pointing towards where the center of the lane will be in d_s meters.

Recalling the generic structure of a controller, as depicted in figure 4.3, we will define our variables in the following way

- y - the output will be given as $y = e_1 + e_2$
- u - the controller output will be the steering wheel angle system input
- r - the reference signal will be zero, since we try to force the error y to be zero

To control the lateral error we will make use of a standard PID controller, which is given by

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \dot{e}(t) \tag{4.63}$$

4.6.2 Longitudinal Controller

A widely spread technology for longitudinal control currently being used is Cruise Control. There are several variants, the simplest one simply keeps the vehicle with a constant velocity. More complex versions make use of a radar system (or an alternative sensor) equipped in the car, and are able to do collision avoidance, braking the car and warning the driver when a car in front

is dangerously close. The more complex Adaptive Cruise Control is even able to adapt the current velocity so as to safely drive behind the preceding vehicle.

For the development of this controller we will assume that we know a reference trajectory, corresponding to a longitudinal position in the road, and that we are also able to measure our current longitudinal position. It is important to note that to our knowledge there are currently no systems providing this kind of measurement directly.

Having a reference longitudinal position given by $x_r(t)$ and a current longitudinal position $x(t)$, we simply compute the longitudinal error as $e_x(t) = x_r(t) - x(t)$.

The control objective is to make this error go to zero, as such we take the same approach as the one taken for the Lateral Controller. The control loop from figure 4.3 is now defined such that

- y - the output will be given as x, the longitudinal position
- u - the controller output will be desired car velocity
- r - the reference signal will be x_r, resulting in $e = x_r - x$

To control this system we will make use, once again, of a PID controller. The desired car velocity that the controller outputs is fed as a reference velocity to the Cruise Control system, which will in turn actuate on the gas/brakes of the car in order to force the car to have that reference velocity.

4.7 Experimental Results

In this section we will test the trajectory tracking performance of the previously explained controllers. The trajectory to be followed consists of an eight shaped lap, as shown in figure 4.8. Each lap has a total duration of $90s$, and a starting point at the intersection, facing the north east direction.

We will spare the reader a report of the tedious parameter tuning task, and will instead only show the final results, corresponding to the best controller parameters found. We also compute the RMS of the errors e_x, e_y and e_θ along the performed trajectory.

4.7.1 Linear Controller

For the linear controller, we found the best parameters k_1, k_2 and k_3 to be

$$\begin{aligned} k_1 &= \frac{\xi}{2}\sqrt{w_r^2 + bv_r^2} \\ k_2 &= \frac{b}{4}|v_r|^2 \\ k_3 &= \xi\sqrt{w_r^2 + bv_r^2} \end{aligned} \tag{4.64}$$

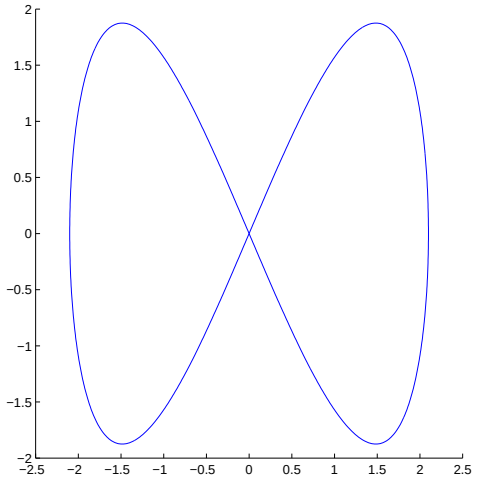


Figure 4.8: Performance testing trajectory

With $\xi = 0.7$ and $b = 60$. These configuration resulted in the trajectory tracking performance shown in figure 4.9a. The computed RMS errors were:

$$x_{\text{rms}} = 0.0648 \quad y_{\text{rms}} = 0.0614 \quad \theta_{\text{rms}} = 0.1143$$

By adding the error dynamics, one can slightly improve the controller performance. Keeping the same configuration as before, and adding the following dynamics:

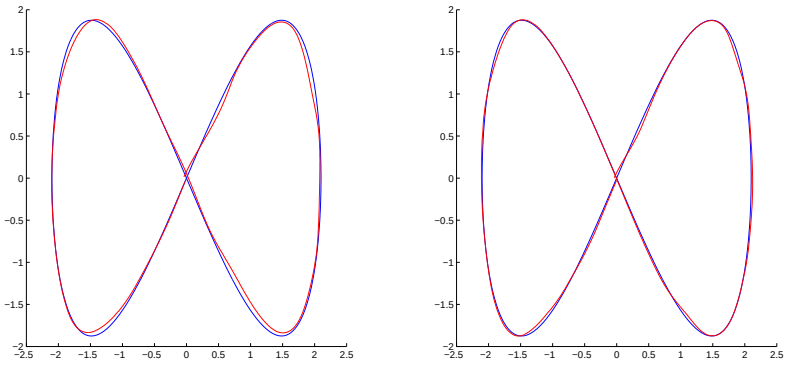
$$\begin{aligned} \bar{e}_y &= K_{p_y} e_y + K_{i_y} \int e_y + K_{d_y} \dot{e}_y \\ \bar{e}_\theta &= K_{p_\theta} e_\theta + K_{i_\theta} \int e_\theta + K_{d_\theta} \dot{e}_\theta \end{aligned} \tag{4.65}$$

Where $K_{p_y} = 1.0$, $K_{i_y} = 0.25$, $K_{d_y} = 5.0$, $K_{p_\theta} = 1.0$, $K_{i_\theta} = 0.0$ and $K_{d_\theta} = 1.0$. Adding these new error dynamics resulted in the trajectory shown in figure 4.9b, and the following errors:

$$x_{\text{rms}} = 0.0273 \quad y_{\text{rms}} = 0.0435 \quad \theta_{\text{rms}} = 0.0699$$

4.7.2 Nonlinear Controller

For the Nonlinear Controller we set the parameters to be



(a) Linear controller without error dynamics performance

(b) Linear controller with error dynamics performance

Figure 4.9

$$\begin{aligned} k_1(v_r(t), \omega_r(t)) &= \xi\sqrt{w_r(t)^2 + bv_r(t)^2} \\ \bar{k}_2 &= \frac{b}{2} \\ k_3(v_r(t), \omega_r(t)) &= \xi\sqrt{w_r(t)^2 + bv_r(t)^2} \end{aligned} \tag{4.66}$$

With $\xi = 0.7$ and $b = 60$. This resulted in the performance seen in figure 4.10 and errors

$$x_{\text{rms}} = 0.0331 \quad y_{\text{rms}} = 0.0254 \quad \theta_{\text{rms}} = 0.0867$$

4.7.3 Sliding Mode Controller

The SMC parameters were set as follows, $k_0 = 0.1$, $k_1 = 5.0$, $k_2 = 1.0$, $p_1 = 1.0$, $p_2 = 1.0$, $q_1 = 0.5$, $q_2 = 0.5$. The signum function was approximated by a threshold of 0.25, this helps remove the chattering effect. The larger the threshold, the smaller the chattering effect, however it will have the downside of a larger tracking error. The signum function is implemented as follows:

$$\text{sgn}(x) = \begin{cases} 1 & \text{if } \geq 0.25 \\ 0 & \text{if } 0.25 < x \geq -0.25 \\ -1 & \text{if } < -0.25 \end{cases}$$

The performance is shown in figure 4.11.

At the beggining of the trajectory there is a large tracking error, due to the states moving quickly to the sliding surface but oscillating around it before stabilizing.

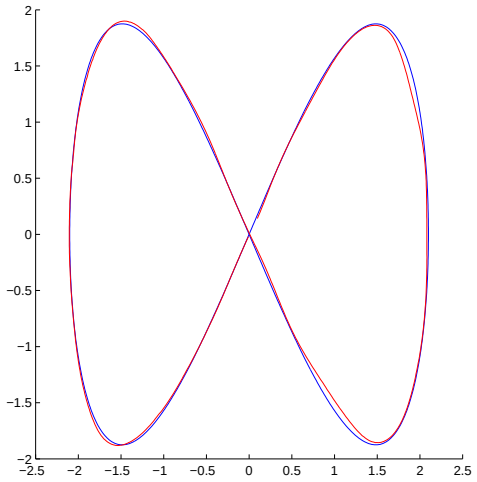


Figure 4.10: Nonlinear controller performance

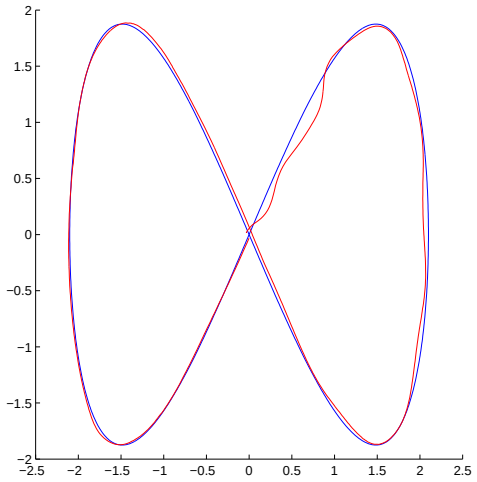


Figure 4.11: SMC performance

The proposed extension of the SMC to the truck trailer case showed a slight improvement in the trajectory tracking, especially in the θ_0 and θ_1 variables, as shown in figure 4.12. However these results were only observed in simulations. In the experimental setup, the proposed extension, behaved worse than the original SMC, and as such the resulting performance is not shown here. It is the opinion of the author, that the addition of another state variable, could make the controller more robust and tolerant to noisy measurements, however it was not possible to prove it experimentally.

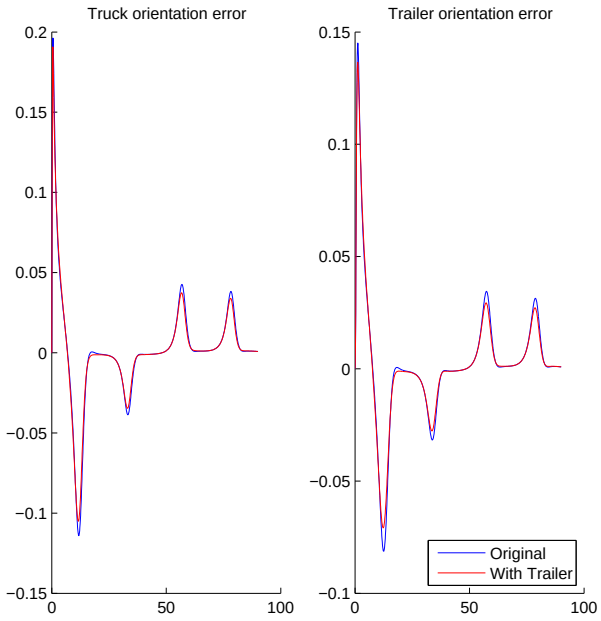


Figure 4.12: SMC extended performance

4.7.4 Decoupled PID Controller

For the Decoupled PID Controller we set a look ahead distance d_s to be 5 metres. The lateral controller PID gains are $K_{p_la} = 10.0$, $K_{i_la} = 1.0$ and $K_{d_la} = 40.0$. For the longitudinal controller we use $K_{p_lo} = 2.0$, $K_{i_lo} = 0.0$ and $K_{d_lo} = 0.0$. The controller performance is shown in figure 4.13.

4.7.5 Discussion and Comparison

Both the Linear and Nonlinear controllers present a small number of parameters to tune, this simplifies greatly the tuning task. The Linear controller was found to have a worse performance than the Nonlinear one. It is possible to

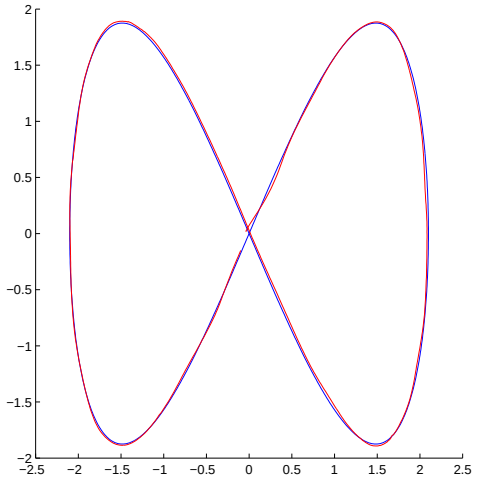


Figure 4.13: Decoupled PID controller performance

make the Linear controller as good as the Nonlinear controller by adding the error dynamics, however this adds extra parameters that need to be tuned.

The Sliding Mode Controller has a large number of parameters, which makes the tuning task tedious and difficult. The SMC has, initially, an erratic behaviour, this happens because the states are converging to the sliding surface, and take a while before stabilizing around them. This behaviour can be seen in the initial section of the trajectory, when the truck is very far away from the desired trajectory. One of the main advantages of general SMCs is their robustness to modelling errors, this can be very important in several situations, including the case of real life trucks where mass and inertia from the cargo and several other parameters can be hard to estimate.

The Decoupled PID Controller has the advantage of having two completely decoupled control objectives, the longitudinal and lateral tracking, this makes the tuning process very easy. Its performance is comparable to the Nonlinear controller, if not slightly better. Also being based on current automotive technologies, it is more suited to real life implementations than the previous controllers. However we must notice that by using it, we are going against the original assumption that we are in unstructured environments, *i.e.* without roads.

There is not a clear winner in thiscomparison, however the performed study allows one to have a clear idea of what to expect from the presented controllers.

Chapter 5

Obstacle Avoidance

In Chapter 2, algorithms were developed which yield a trajectory for the vehicle to follow. This trajectory is assumed safe since it avoids the obstacles known to exist in the environment. The problem to be dealt with in this chapter arises from the possible existence of uncharted obstacles, which cannot be known *a priori*, and fed into the algorithms from Chapter 2.

In section 5.1, the importance of obstacle avoidance is explained, and a brief overview of the sensing technologies is done. Section 5.1 details the approach used to solve the problem, the Model Predictive Controller. In section 5.2 a brief introduction to the topic is given followed by the particular application to the situation we are facing. Section 5.3 closes the chapter by showing the experimental results of the implemented solution.

5.1 Motivation

5.1.1 Unexpected Factors

An autonomous driving system like the one being studied should be able to deal with unpredicted events. In the working environment we will assume the existence of unknown obstacles. These can be the result of several different situations. Some common obstacles that can be present in a previously planned trajectory are parked cars, debris, pedestrians or even wild life. The former two examples consist of static obstacles, *i.e.*, their position is fixed with respect to the environment. The latter two correspond to moving obstacles, which change their position with respect to the environment. To guarantee the correct functioning of the system, the truck should be able to avoid these obstacles, and proceed with its original task, trajectory following, when it becomes once again possible. Figure 5.1 depicts an example of a fixed obstacle.

5.1.2 To Avoid or to Replan

When faced with a previously unknown obstacle, the truck could simply add the recently acquired information of this unknown obstacle to its initial map. It

Figure 5.1: Example of a fixed obstacle, a fallen boulder in the middle of a path

could then start planning a new trajectory, which would successfully avoid this new obstacle. For this purpose the planning algorithm must perform extremely fast, so that it can successfully plan a new trajectory, and for the vehicle to correctly track this new trajectory. The planning algorithms developed in Chapter 2 do not achieve this kind of performance and as such, they are rendered useless for obstacle avoidance (this not implying of course that they are not suited for the initial task of trajectory planning). Note that there the speed limitations come from our particular implementation, there are several examples in the literature of fast RRT planning algorithms being executed in only a few milliseconds. These implementation are referred as to on-line planning (constantly planning possible paths during the execution of a trajectory).

An alternate approach might consist in a simple avoidance technique. A possible solution is to implement a controller, which similarly to the ones detailed in Chapter 4, will try to track a reference trajectory. However it must have a more important purpose, and that would be to avoid obstacles. The controller would simply track the trajectory during normal functioning, and when in the presence of an obstacle, an higher level controller would override the commands, and simply worry about avoiding the obstacle. Once the obstacle is avoided the commands would once again be set by the original trajectory tracking controller, one simple example of this are the *bug algorithms* [54], which simply move towards a goal position, and when facing an obstacle will move away from it.

In the previous paragraphs, we have seen two disparate solutions to the same problem, they can be roughly characterized by replanning and avoiding. Another possible solution to this problem, and a very elegant one, combining replanning and avoiding is found in Model Predictive Control.

A Model Predictive Controller can be seen in part as a planning algorithm, due to its inherent prediction and consideration of future commands and system

states, and also as an avoiding algorithm, since it is able to successfully avoid undesired situations, through the form of constraint handling.

The implemented solution consists of a Model Predictive Controller, which will be explained in detail in section 5.2. This choice reflects the importance of both replanning and obstacle avoidance, and the desire of implementing both.

Model Precitive Control is a well estabilished known technique in Automation Systems. It can also be applied to motion control, [55] and [56] present approaches to control a unicycle system. [57] presents an approach designed for car like vehicles in heavily constrained environments, while [58] makes use of the geometric structure of the road to simplify the problem.

Recently there has been an increasing interest of MPC applied to highly realistic vehicle models. These models are able to include the mass and inertia moments, individual wheel torques and even tire dynamics. These models are often complex, and in order to simplify their implementation, they can be linearised as in [59], [60] and [61]. However, more recent approaches, such as [62] and [63] are solving the original nonlinear model

An extensive introduction to Model Predictive Control can be found in [64].

5.1.3 Sensing Technologies

For an obstacle to be detected perception technologies must be used. Recently the automotive industry has been putting a big effort into the research and development of sensing technologies to be used in driving situations.

One of these technologies is the radar, a detection system that can determine the distance to obstacles, through the use of radio waves. These systems are also commonly used in the Robotics field because of their reliability and accuracy [45].

A radar is able to perceive the presence of an obstacle, up to a certain range. It does so by emitting radio waves, and by sensing the reflections that travel back after colliding with objects. The computation of the distance to the object is often done by measuring the time of flight, in which the distance from the radar to the object and back, can be computed, by knowing the time it took for the echo to return, and the velocity of propagation.

For our purposes we will consider that we have a radar-based sensing system. This system will output the position and boundaries of an obstacle. Since it is radar based, it will only be able to perceive the obstacle when it is closer than a certain range. It will be assumed that when the obstacle is on the radar range, the sensing system will broadcast the boundaries of the obstacle. A schematic of the system is shown in figure 5.2.

5.2 Model Predictive Controller

Model Predictive Control is an approach to Control which combines a model of a system, a prediction of future events and a control objective in order to determine the controlling inputs to be applied to the system. In this section

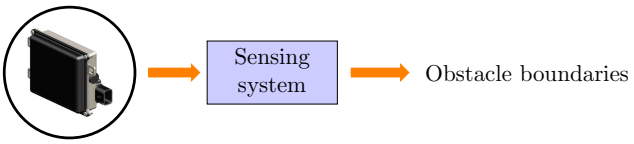


Figure 5.2: Sensing system diagram - receiveing information from radar, and processing it, finally outputting the obstacle boundaries

we will discuss all the fundamental components of the MPC, and how it can be applied to our problem.

5.2.1 Formulation

Model Predictive Control is based on solving an optimization problem of the following form

$$\begin{aligned} \underset{\mathbf{u}}{\text{minimize}} \quad & J(\mathbf{x}, \mathbf{u}) \\ \text{subject to} \quad & x_{k+1} = f(x_k, u_k) \end{aligned} \tag{5.1}$$

Where $\mathbf{u}$ corresponds to the set of inputs to be applied in the future, $\mathbf{x}$ represents the future predicted states of the system, as obtained by the system function $f(\mathbf{x}, \mathbf{u})$.

Once known the optimal $\mathbf{u}$ that minimizes the objective function $J(\mathbf{x}, \mathbf{u})$, the first element of $\mathbf{u}$, corresponding to the immediately next sampling time, is applied to the system, the current state x is measured, and the problem is once again solved, now with new information about the current system state. In the next sampling time, the first element of $\mathbf{u}$ is once again applied. This strategy requires the controller to solve, at each sampling time, an optimization problem.

The formulation in (5.1) is a very general one, to specify it to our system, we rewrite the optimization problem as

$$\begin{aligned} \underset{\mathbf{u}}{\text{minimize}} \quad & \sum_{i=k+1}^{k+n} \|\mathbf{q}_{r_i} - \mathbf{q}_i\| \\ \text{subject to} \quad & \mathbf{q}_1 = f(\mathbf{q}_0, \mathbf{u}_0) \\ & \vdots \\ & \mathbf{q}_n = f(\mathbf{q}_{n-1}, \mathbf{u}_{n-1}) \end{aligned} \tag{5.2}$$

Vector $\mathbf{q}_i$ is defined as the system state at the instant $i \times t_s$, with t_s being the controller sampling time, and corresponding to (x_i, y_i, θ_i). Vector $\mathbf{q}_{r_i}$ is the corresponding reference state, corresponding to the trajectory which the truck should follow.

5.2.2 Vehicle Model

The constraints defined as $\mathbf{q}_t = f(\mathbf{q}_{t-t_s}, \mathbf{u}_{t-t_s})$ encode the nonholonomic constraints corresponding to the discrete version of the vehicle model.

We decided to use the car vehicle model and disregard the trailer, and as such we add the following constraints to the optimization problem.

$$\begin{aligned} x_t &= x_{t-t_s} + t_s V_{t-t_s} \cos(\theta_{t-t_s}) \\ y_t &= y_{t-t_s} + t_s V_{t-t_s} \sin(\theta_{t-t_s}) \\ \theta_t &= \theta_{t-t_s} + t_s \omega_{t-t_s} \end{aligned} \tag{5.3}$$

5.2.3 Input Horizon

Recalling equation (5.2), we notice that the optimization variable only considers the vehicle states corresponding to the summation index i. This index is taken from the immediately next sampling instant, $i = k + 1$, until $i = k + n$. Symbol n is commonly referred to as input horizon. This input horizon corresponds to the number of time instants in which we are optimizing the future input commands. In some cases, usually only solvable through analytic methods, $n = \infty$, those cases fall under the nomenclature of Infinite Horizon Model Predictive Control.

The input horizon is directly related to the predictive characteristic of the controller (similar to a look ahead distance). A larger input horizon would allow the system to predict more into the future, and as such, of being able to react in a more deliberate way to possible obstacles and or changes in the trajectory.

Ideally one would have the maximum possible input horizon n, however that would also imply a large number of optimization variables. Being a problem that must be solved at each sampling time, one must require the problem to be easy enough to be solved under t_s, usually around 0.1 seconds.

Choosing an adequate input horizon n is then a trade-off between desirable large predictive horizon, and the constraint that the problem must actually be solved in a short amount of time.

5.2.4 Output Horizon

A possible extension to the model predictive controller can be made, such that it is possible to drastically increase the prediction horizon, without increasing the complexity associated with a bigger number or optimization variables.

To do so, the concept of output horizon [65] is introduced (not to be confused with the previously discussed input horizon). The output horizon corresponds to the instants of time in which the optimization function is defined, that is the indexes of the summation term in (5.2). We start by redefining the optimization problem with the new variable n_o, corresponding to the output horizon. The input horizon will now be referred to as n_i (previously n). The problem is now formulated as

$$
\begin{aligned}
\underset{(u_0,\ldots,u_{n_i-1})}{\text{minimize}} \quad & \sum_{i=k+1}^{k+n_o} \|\mathbf{q}_{r_i} - \mathbf{q}_i\| \\
\text{subject to} \quad & \mathbf{q}_1 = f(\mathbf{q}_0, \mathbf{u}_0) \\
& \vdots \\
& \mathbf{q}_{n_o} = f(\mathbf{q}_{n_o-1}, \mathbf{u}_{n_o-1}) \\
& u_{n_i} = u_{n_i-1} \\
& \vdots \\
& u_{n_o-1} = u_{n_i-1}
\end{aligned}
$$

Notice that even though the output horizon is n_o, and it is considered in its entirety in the optimization function, the optimization variables are only $\mathbf{u} = (u_0, \ldots, u_{n_i-1})$. The remaining input commands until the output horizon are simply equal to the last command input to be considered as an optimization variable, *i.e.* $u_{n_i} = u_{n_i+1} = \ldots = u_{n_o-2} = u_{n_o-1} = u_{n_i-1}$.

By defining two different horizons, the input horizon and the output horizon, one can achieve a larger predictive horizon while not increasing excessively the optimization performance. The larger predictive horizon comes from using an output horizon that is larger than the input horizon ($n_o > n_i$). By doing so we can extend the time horizon in which we are predicting what is going to happen to the truck, thus resulting in an larger predictive ability of the controller. By keeping the same input horizon n_i we can expect the optimization performance to be similar to the one of problem (5.2) with $n = n_i$, since the number of optimization variables is the same. However there is a small degradation on performance, and this is due to the new terms from $i = n_{i+1}$ until $i = n_o$, which must be taken into account in the optimization.

A common pitfall associated with the use of the input/output horizons is when one defines as excessively large output horizon, when compared to the input horizon. If one does so, one is asking the controller to try and control the truck in a large segment of the trajectory using only a fixed command input (the last optimization variable). This might work well if the trajectory is simply a straight line or an arc, but can behave very poorly if the trajectory is composed of multiple changes of direction.

One might argue that since only the first variable is applied as input to the truck, we should not be worried about what is happening with the last optimization variable, this however, is not true, because the whole optimization process (including the initial optimization variables) can be affected by the events happening in the output horizon.

5.2.5 Move Blocking

Another possible modification in the formulation of the problem consists in including move blocking intervals [66], allowing the algorithm to maintain its predictive horizons while reducing the number of variables.

This is accomplished by the following reformulation

$$\begin{aligned} \underset{(u_0, u_{T-1}, \ldots, u_{n_i-T})}{\text{minimize}} \quad & \sum_{i=k+1}^{k+n_o} \|\mathbf{q}_{r_i} - \mathbf{q}_i\| \\ \text{subject to} \quad & \ldots \\ & u_1 = u_2 = \ldots = u_{T-1} = u_0 \\ & \vdots \\ & u_{n_i-T+1} = u_{n_i-T+2} = \ldots = uu_{n_i-1} = u_{n_i-T} \end{aligned} \tag{5.4}$$

The new constraints added to this version of the algorithm simply encode move blocking intervals, where the input command is constant. The constant T corresponds to the number of samples in which the command is fixed. The optimization variables are the input commands at each T sampling times, the commands in between are simply assumed to be equal to the most recent input variable to be optimized.

By implementing this modification, it is possible to reduce the number of optimization variables by a factor of $\frac{1}{T}$. This comes with an immediate performance improvement, as it will now run in a much faster time. The gain in performance is done without shortening the predictive horizon, thus representing a very desirable improvement.

One can recall the problems arising from a much larger output horizon as compared to input horizon (recall section (5.2.4)), and imagine them happening also if the move blocking interval is chosen to be very large. In a situation where this interval is too large, one can deal with problems of trying to perform a complex manoeuvre, while just optimizing one fixed input command (corresponding to the current blocking interval). This can of course result in catastrophic results, and as such the interval must be chosen with care.

One might assume that this approach is equivalent to increasing the sampling time by a factor of T, however this is not true for the reasons stated next. If one were to increase the sampling time, the path approximations made by the discretized vehicle model would be coarser. The lower the sampling time, the more precise the vehicle path (if the sampling time tends to zero, the discretized model becomes equivalent to the continuous model). Figure 5.3 illustrates both path approximations.

Besides the rougher path approximation, the choice of increasing the sampling time would also result in a less responsive controller, as it would not be able to quickly react to possible perturbation (be it obstacles, or simply tracking errors).

5.2.6 Obstacle avoidance

The obstacle avoidance behaviour can be easily implemented in the MPC through the formulation of constraints. One can simply add a new set of con-

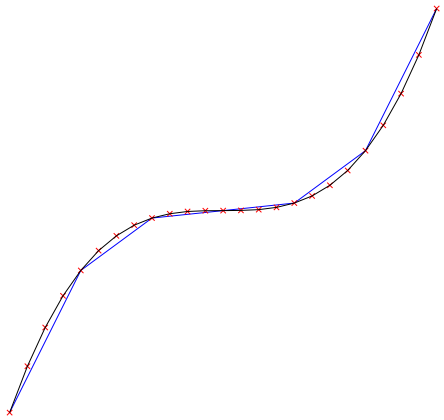


Figure 5.3: Path approximation in the case of move blocking (black) and in the case of a larger, by a factor of four, sampling time (blue)

straints to the optimization problem, such that they force the vehicle to be outside of the obstacle region $\mathbf{X}_{obs}$.

$$\begin{aligned} \underset{\mathbf{u}}{\text{minimize}} \quad & \sum_{i=k+1}^{k+n} \|\mathbf{q}_{r_i} - \mathbf{q}_i\| \\ \text{subject to} \quad & \dots \\ & (x_1, y_1) \notin \mathbf{X}_{obs} \\ & \vdots \\ & (x_n, y_n) \notin \mathbf{X}_{obs} \end{aligned}$$

The new formulation implies that the point (x, y) cannot be inside the obstacle region $\mathbf{X}_{obs}$, in any moment of time $i = k + 1, \dots, k + n$.

We are only concerned here that the center of the vehicle, given by (x, y) is outside of the obstacle region. This however does not imply that the vehicle will not collide, since it has a width and a length, the solution here, is to expand the true obstacle region, such that it accounts for the dimensions of the vehicle. Therefore our region $\mathbf{X}_{obs}$, will not correspond to the true obstacle location and region, but instead to an expanded version of it. $\mathbf{X}_{obs}$ will then correspond to the obstacle enlarged by the half the width of the vehicle, in this way we are forcing the center of the vehicle to always be at a safe distance from the obstacle.

5.2.7 Trajectory Tracking Tuning

It is possible to attribute different weights to the several components of the error function. In the previous optimization problem formulations, the objective function was based on the norm given by $\|\mathbf{q}_{r_i} - \mathbf{q}_i\|$. However a more complex distance function can be used:

$$K_x|\mathbf{x}_{r_i} - \mathbf{x}_i| + K_y|\mathbf{y}_{r_i} - \mathbf{y}_i| + K_\theta|\theta_{r_i} - \theta_i|$$

This distance function allows the designer to give more emphasis to a specific error. By having K_x and K_y bigger than K_θ one can force the controller to track the position more precisely, at the cost of a bigger angle displacement.

The choice of K_x, K_y and K_θ is not an obvious one. One should try several combinations, and keep one yielding the best results.

5.2.8 Beyond Trajectory Tracking

The previous formulations of the optimization problem only seek to minimize the error of the truck state to the reference trajectory. It is possible, however to add new terms to the objective function J to be minimized.

A possible reformulation consists in adding smoothness terms that penalize fast changes in input commands. One way to do it, is through the following formulation

$$\begin{aligned} \underset{\mathbf{u}}{\text{minimize}} \quad & \bar{J} + \sum_{i=k+1}^{k+n-1} K_V\|V(i) - V(i-1)\| + K_\omega\|\omega(i) - \omega(i-1)\| \\ \text{subject to} \quad & \ldots \end{aligned} \tag{5.5}$$

$\bar{J}$ corresponds to the original formulation of the objective function, and is a measure of the tracking error. The new terms add a penalization for changes between consecutive input commands. V and ω are the control inputs, corresponding to the linear and angular velocity, respectively. The constants K_V and K_ω are weighting factors which decide the relative importance of the smoothness of the velocity and steering commands respectively.

If one chooses very large weighting factors, the optimization problem will put more effort into smoothing the controls, which might eventually result in a worsening of the trajectory tracking. By choosing small weighting factors, one would have the opposite effect, of having a better tracking, but not putting a lot of effort into having smooth control commands. The choice of K_V and K_ω is then not obvious, and several combinations must be tested in order to choose the best possible.

5.2.9 Dynamics Modelling

As shown in Chapter 3, the mini truck actuators have an undesirable delay which severely affects controller performance. If one can model this actuator

behaviour, one can also try to compensate for it, by simply including it in the MPC formulation. One can do so by adding constraints that force the optimization problem to take into account the dynamics.

We will introduce the models that were used to try to include the dynamical behaviour of the actuators in the MPC.

Pure delay model

The pure delay model simply assumes that the actuator dynamics correspond to a pure delay. This assumption is based on the experience communication delays, between the computer and the truck, which can be easily modelled through a pure delay. We assume that the system can be modelled by the following discrete function

$$y(k) = u(k - \bar{\tau}) \tag{5.6}$$

Where y corresponds to the system output and u to the system input. The constant $\bar{\tau}$ corresponds to the actuator delay, and it is chosen to mimic as best as possible the true system.

We define the linear velocity actuator delay to be $\bar{\tau}_v = 0.7$ and for the steering actuator $\bar{\tau}_\phi = 0.4$. Both delays should be the same, however we found that this combination of delays provided the best results. This is due to the fact, that the delay is a result of a communication delay, and an actuator delay. The actuator delay is larger in the linear velocity case, thus resulting in a $\bar{\tau}_v$ larger than $\bar{\tau}_\phi$. We are effectively adjusting our pure delay model, such that it also takes into account the actuator dynamics.

Discretized first order system model

Another possible way to model the actuator characteristics is through a first order system model. We do so by using a discretized first order system model. Notice that we are now ignoring the time delays resulting from communication times.

A first order system with a unit gain is given by

$$G(s) = \frac{1}{\tau s + 1} \tag{5.7}$$

We assume that the gain is unitary since we are only interesting in modelling the delay in the MPC. To actually apply the correct input such that the gain of the actuator is taken into account, one must simply multiply the optimal input found by solving the optimization problem by the inverse gain.

Assuming a Zero Order Hold approximation, that is, the input commands are fixed between sampling times, we can compute the discretized model as

$$G(z) = \frac{Y(z)}{U(z)} = \left(1 - z^{-1}\right) \mathcal{Z}\left\{\mathcal{L}^{-1}\left\{\frac{G(s)}{s}\right\}\right\} \tag{5.8}$$

Yielding

$$(z - e^{-\frac{T}{\tau}})Y(z) = (1 - e^{-\frac{T}{\tau}})U(z) \tag{5.9}$$

Where T stands for the sampling time. Replacing $U(z)$ by $u(k)$, $Y(z)$ by $y(k)$ and $zY(z)$ by $y(k-1)$ we get

$$y(kT) = \frac{y((k-1)T) - (1 - e^{-\frac{T}{\tau}})u(kT)}{e^{-\frac{T}{\tau}}} \tag{5.10}$$

Equation (5.10) determines the actuator output, based on the current input and the previous output. This equation is added as a constraint in the MPC formulation.

Acceleration based model

The acceleration based model is only concerned about modelling the linear velocity actuator dynamics. The steering actuator is assumed to be without delay, and only the linear velocity actuator will be subject to additional constraints in the formulation of the optimization problem.

The idea for this model stems from the assumption that the most important non linearities of the velocity actuator are the acceleration limits. We assume that there is no delay in the actuator, that there is no wheel slippage and that air drag forces are insignificant, thus resulting in creating a model where only the maximum acceleration (or braking) limit the rate at which the velocity can be changed.

To formulate this model we simply write

$$v((k+1)T) = \begin{cases} v(kT) - T \times b_{max} & \text{if } v^*((k+1)T) \leq v(kT) - T \times b_{max} \\ v(kT) + T \times a_{max} & \text{if } v^*((k+1)T) \geq v(kT) + T \times a_{max} \\ v^*((k+1)T) & \text{otherwise} \end{cases} \tag{5.11}$$

Where $v^*((k+1)T)$ represents the desired velocity, and $v((k+1)T)$ the actual achievable velocity, as limited by the maximum acceleration, a_{max}, and maximum brake, $b_{max} > 0$, constraints.

5.3 Experimental Results

5.3.1 Setup

In order to test the performance of the MPC controller and its different models, an experiment was set-up in the laboratory. The vehicle will only be composed of the mini truck, the trailer will not be used.

The trajectory to be performed is a circle, where each lap is to be completed in 45 seconds, resulting in a constant linear speed. The trajectory equation is given as

$$x(t) = \cos\left(\frac{2\pi}{45}t\right) \qquad y(t) = \sin\left(\frac{2\pi}{45}t - \frac{\pi}{2}\right) \tag{5.12}$$

We simulate an obstacle, given by the circle equation, and with a radius of 0.125 metres.

$$(x - x_{obs})^2 + (y - y_{obs})^2 \leq 0.125^2 \tag{5.13}$$

To simulate the sensing characteristics of the radar, the constraints corresponding to the obstacle avoidance ((5.2.6)), are only added when the mini truck front is at a distance of 2.5 meter from the object. This way we can simulate the range of the radar, as a cone centered in the front bumper of the mini truck, as seen in figure 5.4.

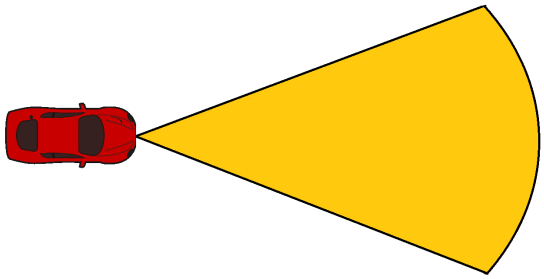


Figure 5.4: The sensing cone of the radar system. Only obstacles in the yellow region are detected

To simplify the task at hand, we assume that once the obstacle region intersects the yellow cone, the whole obstacle region $\mathbf{X}_{obs}$ is known, and only then is it added to the MPC constraints.

5.3.2 MPC Implementation

To solve the optimization problem in which the MPC consists, we used MATLAB's function `fmincon`. One can use such function with the following syntax:

`fmincon(obj_fcn,commands_0,A,b,Aeq,beq,lb,ub,non_l_con)`

`obj_fcn` is the function corresponding to the objective function. It simply returns the value of the function in equation (5.5).

The arguments *Aeq* and *beq* are left empty, because we do not need to use equality constraints. Arguments A and b are such that they limit the commands to a maximum angular and linear speed magnitude. *lb* and *ub* are left empty. *non_l_con* is a function to force non-linear constraints. In our implementation the function simply returns the vector

$$\mathbf{c} = r_{obs}^2 - (\mathbf{x} - x_{obs})^2 - (\mathbf{y} - y_{obs})^2 \tag{5.14}$$

`fmincon` will then ensure that $\mathbf{c} < 0$, and as such, that $(\mathbf{x}-x_{obs})^2+(\mathbf{y}-y_{obs})^2 > r^2$, thus guaranteeing that all of the states are outside of the obstacle zone corresponding to the circle of radius r.

The function options are specified such that the solving algorithm used is `'active-set'`.

5.3.3 Trajectory Tracking

In order to study the performance of the MPC we will start by evaluate its trajectory tracking ability. The trajectory in study is still the circle mentioned in 5.3.1, however without the presence of the obstacle.

Pure delay model

A full lap is done in order to study the performance of the MPC for different values of K_ω, as it is used in equation (5.5). Table 5.1 presents several performance indices for different parameter combinations. RMSE stands for the root-mean-square error of the trajectory tracking objective. The Steering Stress corresponds to the sum of the square of the differences between consecutive steering angle commands, it is thus a measure of how much steering done. Here we consider that a constant steering, even if high, corresponds to a null Steering Stress, the objective of this measure is not to measure the steering directly, but how much it changes during the driving task. A lower steering stress is usualy associated with a safer and more confortable driving experience.

These performance indices were taken for several combinations of the size of move blocking intervals (MB), input horizon (IH) and output horizon (OH), and also different values of the smoothing parameter K_ω.

MB	IH	OH	Smoothing K_ω	RMSE			Steering Stress
				x	y	θ	
			0	0.0565	0.0165	0.0899	30.0030
4	16	32	0.5	0.0525	0.0084	0.0260	0.0751
			2.0	0.0520	0.0112	0.0271	0.0219
			0	0.0838	0.0145	0.0696	14.1268
6	24	48	0.5	0.0609	0.0073	0.0323	0.0308
			2.0	0.0695	0.0078	0.0396	0.0135
			0	0.0644	0.0173	0.0830	13.6665
6	30	60	0.5	0.0825	0.0082	0.0477	0.0170
			2.0	0.0546	0.0078	0.0317	0.0136
			0	0.0968	0.0097	0.0588	0.1415
8	32	64	0.5	0.0958	0.0099	0.0589	0.0135
			2.0	0.0906	0.0111	0.0552	0.0143

Table 5.1: Performance of the MPC with pure delay model for diverse parameter combinations

Figure 5.5 shows the different steering commands issued by the controller, for different values of K_ω, for a particular combination of move blocking interval, input horizon and output horizon. This figure shows the importance of having a non zero K_ω, in order to avoid an highly oscillatory behaviour of the steering command.

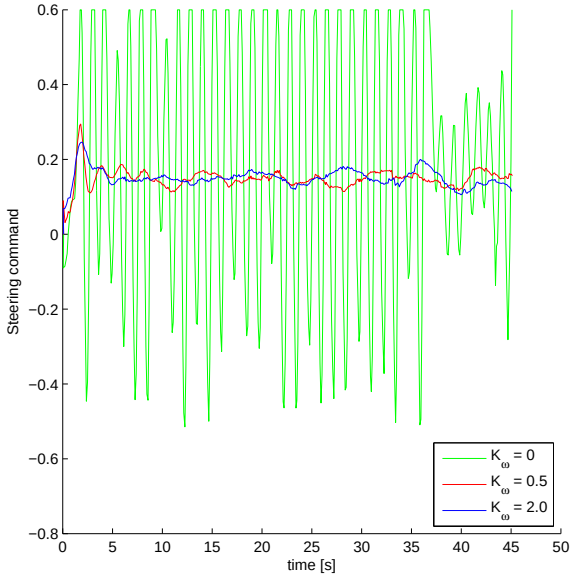


Figure 5.5: The influence of the smoothing parameter K_ω on the steering command behaviour

Looking at table 5.1, we can see that for values of $K_\omega = 0.5$ or $K_\omega = 2.0$, the steering stress is low. The trajectory tracking error is small, and similar between different configurations. The configuration chosen as the best corresponds to having a a Move Blocking interval of 4, Input Horizon of 16 and Output Horizon of 32, since its performance was good, and the optimization algorithm was solvable in a time safely smaller than the sampling time.

Discretized first order model

The experimental set-up is the same as before. For this vehicle model, the optimization problem takes longer to solve. In order to be able to use this formulation of the optimization function, it becomes necessary to increase the controller sampling time from $0.1s$ to $0.2s$.

Table 5.2, shows the performance of this MPC.

The performance indices are similar to the ones obtained for the pure delay model, that is, the RMSE values are of similar magnitude. One important

MB	IH	OH	Smoothing K_ω	RMSE x	RMSE y	RMSE θ	Steering Stress
			0	0.0659	0.0086	0.0495	0.1307
4	12	24	0.5	0.0500	0.0065	0.0364	0.0253
			2.0	0.0791	0.0078	0.0556	0.0244
			0	0.0849	0.0059	0.0582	0.1741
4	16	32	0.5	0.0835	0.0122	0.0619	0.0305
			2.0	0.0709	0.0087	0.0481	0.0176
			0	0.0867	0.0115	0.0621	0.0507
6	18	36	0.5	0.0750	0.0159	0.0578	0.0189
			2.0	0.0938	0.0073	0.0637	0.0166

Table 5.2: Performance of the MPC with discretized first order model for diverse parameter combinations

point to notice is the steering stress, which is much lower when $K_\omega = 0$. A possible reason for this might consist in the very precise vehicle model that can be achieved using the first order systems to model the actuator delays.

Even though this MPC outperformed the previous model, it will not be used here after, due to the requirement of an increased sampling rate, which can result in dangerous manoeuvres when facing obstacles. As an example, if we have a very low sampling rate, we might get the knowledge of the obstacle only when we are already too close to it. Having a very high sampling rate ensures that the obstacle is accounted for almost as soon as when it is in the sensing cone.

Acceleration based model

The values experimentally found for the maximum acceleration and braking rates were $a_{max} = 0.4102$ and $b_{max} = 0.9009$. The sampling time is once again set to $0.1s$. Table 5.3 shows the performance of this MPC.

We can notice that the orientation error is significantly larger when compared to the previous MPCs. The reason for this might be related to the fact that only the dynamics of the velocity actuator are here modelled, while the steering dynamics are not considered. This model will also be disregarded here after, since we can achieve a better performance with the pure delay model.

5.3.4 Obstacle Avoidance

After some testing and benchmarking, we decided to choose the pure delay MPC with parameters Blocking = 6, Input Horizon = 24 and Output Horizon = 48. We chose this due to its consistently good performance and short solving time.

MB	IH	OH	Smoothing K_ω	RMSE x	RMSE y	RMSE θ	Steering Stress
			0	0.0452	0.0279	0.1815	8.3127
4	16	32	0.5	0.0322	0.0108	0.4265	0.0950
			2.0	0.0365	0.0135	0.5218	0.0373
			0	0.0529	0.0107	0.8469	2.5357
6	24	48	0.5	0.0493	0.0112	0.3042	0.1109
			2.0	0.0523	0.0148	0.0384	0.0267
			0	0.0529	0.0107	0.8469	2.5357
6	30	60	0.5	0.0493	0.0112	0.3042	0.1109
			2.0	0.0523	0.0148	0.0384	0.0267
			0	0.0723	0.0098	0.6636	0.4781
8	32	64	0.5	0.0698	0.0154	0.0479	0.2557
			2.0	0.0644	0.0201	0.0495	0.0520

Table 5.3: Performance of the MPC with acceleration based model for diverse parameter combinations

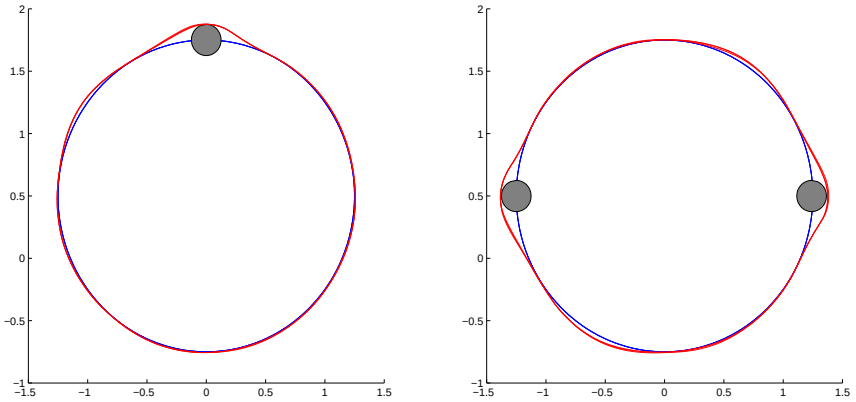
Static Obstacles

Figure 5.6a shows the performance of the MPC, when facing a static obstacle located at the top of the trajectory. Three consecutive laps were performed, and in all three of them the obstacle was successfully avoided. It is important to notice the smoothness of the avoidance, and the consistency of the performed avoidance manoeuvre between different laps, indicating that the MPC is stable and predictable. One can notice that after the avoidance there is a slight trajectory tracking error (in the north west section of the trajectory), that takes a while to be recovered. This happens due to the value of K_ω, which has to be high enough to guarantee a smooth steering behaviour.

Figure 5.6b shows the same MPC, when facing two static obstacles, located at the west and east sections of the trajectory. Three laps were performed, and the obstacles were avoided with success. The remarks made before also apply to this test.

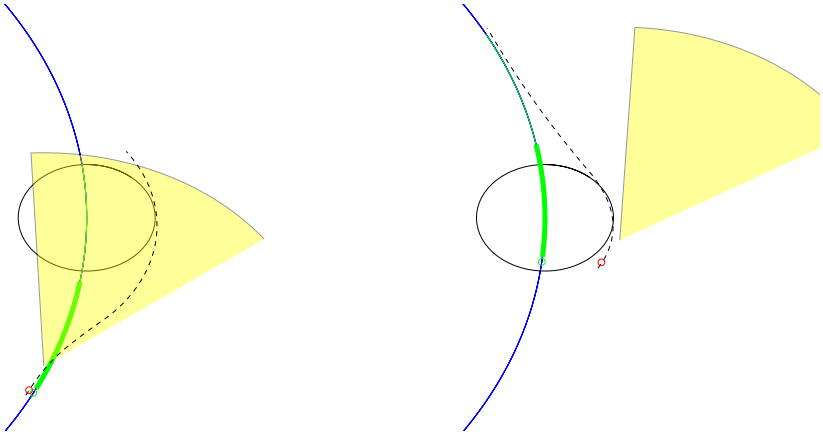
Figure 5.7 shows in more detail the obstacle avoidance. The red dot corresponds to the rear axle position of the truck. In blue we show the desired trajectory, and in green we show the desired trajectory during the prediction horizon. The thick green section corresponds to the input horizon, while the thin green section corresponds to the output horizon. The yellow cone represents the radar sensing range, and the dashed black line is the predicted optimal trajectory resulting from solving the optimization problem in the MPC.

Figure 5.7a shows the truck approaching the obstacle and planning an avoidance manoeuvre, while figure 5.7b shows the truck already contouring the obstacle, and planning a merge with the original trajectory. Note that even though the obstacle is outside the sensing cone, it is still assumed known, since it was previously detected.



(a) Avoiding one static obstacle (b) Avoiding two static obstacles

Figure 5.6



(a) Approaching the obstacle (b) Contouring the obstacle

Figure 5.7

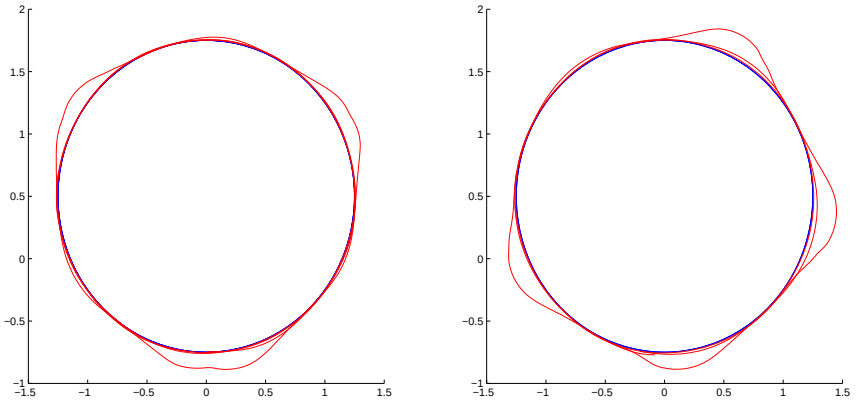
Moving Obstacles

In figure 5.8a, we show the performance of the MPC when facing a moving obstacle. This obstacle is on top of the trajectory, performing it with a speed equal to $\frac{1}{4}$ of the desired truck speed. The obstacle is moving in the same direction of the truck, counter clockwise. Four laps are done, and the obstacle is successfully avoided in the three times it enters the truck's way.

The avoidance manoeuvre is slightly longer than than the one for a moving obstacle, and can sometimes be more aggressive, as can be seen in the south region of the trajectory, where the truck had to re-steer in order not collide with the obstacle. The not so smooth behaviour noticed here, is related to the optimization problem not being able to be solved under the $0.1s$, and as such aborting, resulting in a suboptimal solution.

Figure 5.8b, shows the performance for a moving obstacle, however, now in a clockwise movement, that is, moving towards the truck. When the truck starts facing the obstacle, the MPC begins a steering movement to avoid it. However, due to its approaching movement, which is not taken into account in the MPC formulation, the avoiding manoeuvre is sometimes not enough to avoid the obstacle.

In this experiment the obstacle was successfully avoided in two of the four crossings. In the two collisions the truck touched the obstacle slightly, that is, it was not a full collision, but only a scraping with the obstacle. This could be solved either by artificially expanding the obstacle bounds, or by implementing the obstacle dynamics in the MPC formulation.



(a) Avoiding a forward moving obstacle (b) Avoiding a backward moving obstacle

Figure 5.8

Chapter 6

Conclusion

The work developed provided a great insight into the possible automation of driving tasks in HDV, as is the objective of the iQMatic project.

The planning algorithms studied proved to be suitable for the particular vehicle model in study, a HDV consisting of a truck and a trailer, and to the types of environments that we are expecting to encounter, unstructured and heavily cluttered.

Future work to be done in the planning part, if such would be required, would consist of implementing the algorithm in a faster programming language, such as `C`. A deeper study into possible heuristic distance metrics is also suitable, since they present a very important part of the algorithm, influencing both the performance and final result. Including reverse manoeuvres might become necessary if the environment is expected to be very complex, however it is not obvious how to add such a feature without degrading substantially the algorithm performance, and as such, one must approach the problem with care. The proposed genetic algorithm opens many possibilities for future work, as one could delve deeper into the topic, and implement more complex techniques in order to improve its performance. Cost functionals other than simply the path length can also be tried, some possible alternative cost functionals would consist of travel time, fuel efficiency or even a mix of both.

The testing and comparison of the controllers gave us an idea of their performance. Both linear and nonlinear controllers showed themselves to be good in the trajectory tracking, while the SMC failed to meet our performance requirements. The market controller had a great performance however we must be reminded that it is only able to function in roads. We expect that controller performance would increase when applied to real life trucks, since we would be able to do a more precise control, specifically in the velocity and steering actuators, which are not directly measurable in the SML, but that would be in real life. In the real trucks however, we would be requires higher velocities, at which the kinematic model starts being highly unfaithful, due to the presence of dynamics. In these cases, more sophisticated controllers must be used, which take into account wheel torques, moments of inertia of the vehicle and even road friction.

The implemented MPC techniques showed a great trajectory tracking performance, whilst being able to avoid possible obstacles. The MPC proved to be as good as the best controllers tested in the trajectory tracking section. However, due to time constraints, we were not able to perform a deep study and development of the obstacle avoidance, and as such the MPC presents many possible improvements. Including moving obstacles, design a possible trajectory linearisation in order to simplify the optimization problem or use/develop specialised numerical solvers for the optimization problem are only a few of the possible improvements. When applied to real truck, one must take into account that the kinematic model becomes useless and as such better models must be used.

This thesis has successfully made a study on the feasibility of autonomous driving in HDV, while also laying ground work for possible future research. Some interactive demonstrations were also developed and are now available for exhibition, showing how an autonomous driving system works, these demonstrations are both suitable for knowledgeable people in the automation area and for people without any kind of engineering education.

Chapter 7

Bibliography

[1] Susan M. Coughlin. Pressing issues: Truck safety and driver fatigue. *Consumers' Research Magazine*, 81(11):10, 1998.

[2] Richard Bishop. Intelligent vehicle applications worldwide. *Intelligent Systems and their Applications, IEEE*, 15(1):78–81, 2000.

[3] Pedro F. Lima. Implementation and analysis of platoon catch-up scenarios for heavy duty vehicles. Master's thesis, KTH, Automatic Control, 2013.

[4] João Pedro Alvito. Implementation of traffic control with heavy duty vehicle anti-platooning. Master's thesis, KTH, Automatic Control, 2013.

[5] Francisco Martucci Mejía. On-board recursive state estimation for dead-reckoning in an autonomous truck. Master's thesis, KTH, Automatic Control, 2014.

[6] Meng Guo. Cooperative motion and task planning under temporal tasks, 2014. QC 20140225.

[7] Diogo Almeida. Event-triggered attitude stabilization of a quadcopter. Master's thesis, KTH, Automatic Control, 2014.

[8] Johanna Orihuela Swartling. Circumnavigation with a group of quadrotor helicopters. Master's thesis, KTH, Automatic Control, 2014.

[9] Steven M LaValle. Motion planning. *Robotics and Automation Magazine, IEEE*, 18(1):79–89, 2011.

[10] Steven Michael LaValle. *Planning algorithms*. Cambridge university press, 2006.

[11] OJ Sordalen. conversion of the kinematics of a car with n trailers into a chained form. Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on, 1993.

[12] Lester E Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, pages 497–516, 1957.

[13] Steven M LaValle. Rapidly-exploring random trees a new tool for path planning. 1998.

[14] James Bruce and Manuela Veloso. Real-time randomized path planning for robot navigation. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2383–2388. IEEE, 2002.

[15] Romain Pepy, Alain Lambert, and Hugues Mounier. Path planning using a dynamic vehicle model. In *Information and Communication Technologies, 2006. ICTTA'06. 2nd*, volume 1, pages 781–786. IEEE, 2006.

[16] Th Fraichard and A Lambert. Planning safe paths for nonholonomic car-like robots navigating through computed landmarks. In *Int. Conf. on Intelligent Autonomous Systems*, pages 447–454, 2000.

[17] Stephen R Lindemann and Steven M LaValle. Steps toward derandomizing rrts. In *Robot Motion and Control, 2004. RoMoCo'04. Proceedings of the Fourth International Workshop on*, pages 271–277. IEEE, 2004.

[18] Steven M LaValle and James J Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. 2000.

[19] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28(8):933–945, 2009.

[20] Long Han, Quoc Huy Do, and Seiichi Mita. Unified path planner for parking an autonomous vehicle based on rrt. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5622–5627. IEEE, 2011.

[21] Steven M LaValle and James J Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[22] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.

[23] Peng Cheng and Steven M. Lavalle. Reducing metric sensitivity in randomized trajectory design. In *In IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 43–48, 2001.

[24] S. Karaman and E. Frazzoli. Optimal kinodynamic motion planning using incremental sampling-based methods. In *IEEE Conference on Decision and Control (CDC)*, Atlanta, GA, December 2010.

[25] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[26] Scott Teuscher. Dubins curve mex. `mathworks.se/matlabcentral/fileexchange/40655-dubins-curve-mex`.

[27] Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244–256, 1972.

[28] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

[29] A Pedro Aguiar, Dragan B Dacic, Joao P Hespanha, and Petar Kokotovic. Path-following or reference-tracking? *rn*, 1:1, 2004.

[30] Gregor Klančar and Igor Škrjanc. Tracking-error model-based predictive control for mobile robots in real time. *Robotics and Autonomous Systems*, 55(6):460–469, 2007.

[31] F Kühne, J Gomes, and W Fetter. Mobile robot trajectory tracking using model predictive control. In *II IEEE latin-american robotics symposium*, 2005.

[32] Ivan Maurovic, Mato Baotic, and Ivan Petrovic. Explicit model predictive control for trajectory tracking with mobile robots. In *Advanced Intelligent Mechatronics (AIM), 2011 IEEE/ASME International Conference on*, pages 712–717. IEEE, 2011.

[33] Kiattisin Kanjanawanishkul and Andreas Zell. Path following for an omnidirectional mobile robot based on model predictive control. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3341–3346. IEEE, 2009.

[34] Takanori Fukao, Hisashi Nakagawa, and Norihiko Adachi. Adaptive tracking control of a nonholonomic mobile robot. *Robotics and Automation, IEEE Transactions on*, 16(5):609–615, 2000.

[35] A Keymasi Khalaji and S Ali A Moosavian. Robust adaptive controller for a tractor–trailer mobile robot.

[36] John Arvanitakis, George Nikolakopoulos, Demetris Zermas, and Anthony Tzes. On the adaptive performance improvement of a trajectory tracking controller for non-holonomic mobile robots. In *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*, pages 1–7. IEEE, 2011.

[37] Bihua Chen and Zongxia Jiao. Adaptive path following control of car-like mobile robot using dynamic model. In *Industrial Electronics and Applications (ICIEA), 2011 6th IEEE Conference on*, pages 239–244. IEEE, 2011.

[38] Mišel Brezak, Ivan Petrovic, and Nedjeljko Peric. Experimental comparison of trajectory tracking algorithms for nonholonomic mobile robots. In *Industrial Electronics, 2009. IECON'09. 35th Annual Conference of IEEE*, pages 2229–2234. IEEE, 2009.

[39] M Sampei, T Tamura, T Itoh, and M Nakamichi. Path tracking control of trailer-like mobile robot. In *Intelligent Robots and Systems' 91.'Intelligence for Mechanical Systems, Proceedings IROS'91. IEEE/RSJ International Workshop on*, pages 193–198. IEEE, 1991.

[40] Maciej Michałek. Tracking control strategy for the standard n-trailer mobile robot–a geometrically motivated approach. In *Robot Motion and Control 2011*, pages 39–51. Springer, 2012.

[41] Jorge L Martínez, Jesús Morales, Anthony Mandow, and Alfonso García-Cerezo. Steering limitations for a vehicle pulling passive trailers. *Control Systems Technology, IEEE Transactions on*, 16(4):809–818, 2008.

[42] TR Ren, NM Kwok, C Sui, D Wang, Jiman Luo, and Weidong Su. Controller design of a truck and multiple trailer system. In *Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on*, pages 294–299. IEEE, 2010.

[43] Alessandro Astolfi, Paolo Bolzern, and A Locatelli. Path-tracking of a tractor-trailer vehicle along rectilinear and circular paths: a lyapunov-based approach. 2004.

[44] Claudio Altafini, Alberto Speranzon, and Karl Henrik Johansson. Hybrid control of a truck and trailer vehicle. In *Hybrid Systems: Computation and Control*, pages 21–34. Springer, 2002.

[45] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[46] Alessandro De Luca, Giuseppe Oriolo, and Marilena Vendittelli. Control of wheeled mobile robots: An experimental overview. In *Ramsete*, pages 181–226. Springer, 2001.

[47] C Canudas De Wit, H Khennouf, C Samson, and OJ Sordalen. *Nonlinear control design for mobile robots*, volume 11. Singapore: World Scientific, 1993.

[48] I Barbalat. Systemes d'équations différentielles d'oscillations non linéaires. *Rev. Math. Pures Appl*, 4(2):267–270, 1959.

[49] K David Young, Vadim I Utkin, and Umit Ozguner. A control engineer's guide to sliding mode control. *Control Systems Technology, IEEE Transactions on*, 7(3):328–342, 1999.

[50] Yuri Shtessel, Christopher Edwards, Leonid Fridman, and Arie Levant. *Sliding mode control and observation*. Springer, 2014.

[51] Razvan Solea and Urbano Nunes. Trajectory planning and sliding-mode control based trajectory-tracking for cybercars. *Integrated Computer-Aided Engineering*, 14(1):33–47, 2007.

[52] Razvan Constantin Solea. Sliding mode control applied in trajectory-tracking of wmrs and autonomous vehicles. 2009.

[53] Rajesh Rajamani. *Vehicle dynamics and control*. Springer, 2011.

[54] Howie M Choset. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[55] Heonyoung Lim, Yeonsik Kang, Changwhan Kim, Jongwon Kim, and Bum-Jae You. Nonlinear model predictive controller design with obstacle avoidance for a mobile robot. In *Mechtronic and Embedded Systems and Applications, 2008. MESA 2008. IEEE/ASME International Conference on*, pages 494–499. IEEE, 2008.

[56] Colin Jones, Andrea Alessandretti, A Pedro Aguiar, et al. Trajectory-tracking and path-following controllers for constrained underactuated vehicles using model predictive control. In *European Control Conference 2013*, number EPFL-CONF-186264, 2013.

[57] Yongsoon Yoon, Jongho Shin, H Jin Kim, Yongwoon Park, and Shankar Sastry. Model-predictive active steering and obstacle avoidance for autonomous ground vehicles. *Control Engineering Practice*, 17(7):741–750, 2009.

[58] Julia Nilsson, Mohammad Ali, Paolo Falcone, and Jonas Sjöberg. Predictive manoeuvre generation for automated driving. In *16th International IEEE Annual Conference on Intelligent Transportation Systems*, 2013.

[59] Valerio Turri, Ashwin Carvalho, Hongtei Tseng, Karl Henrik Johansson, and Francesco Borrelli. Linear model predictive control for lane keeping and obstacle avoidance on low curvature roads. 2013.

[60] Paolo Falcone, Francesco Borrelli, H Eric Tseng, Jahan Asgari, and Davor Hrovat. Low complexity mpc schemes for integrated vehicle dynamics control problems. In *9th International Symposium on Advanced Vehicle Control (AVEC'08)*, 2008.

[61] Paolo Falcone, Francesco Borrelli, Jahan Asgari, Hongtei Eric Tseng, and Davor Hrovat. Predictive active steering control for autonomous vehicle systems. *Control Systems Technology, IEEE Transactions on*, 15(3):566–580, 2007.

[62] Janick V Frasch, Andrew Gray, Mario Zanon, Hans Joachim Ferreau, Sebastian Sager, Francesco Borrelli, and Moritz Diehl. An auto-generated nonlinear mpc algorithm for real-time obstacle avoidance of ground vehicles. In *Control Conference (ECC), 2013 European*, pages 4136–4141. IEEE, 2013.

[63] Yiqi Gao, Theresa Lin, Francesco Borrelli, Eric Tseng, and Davor Hrovat. Predictive control of autonomous ground vehicles with obstacle avoidance on slippery roads. In *ASME 2010 Dynamic Systems and Control Conference*, pages 265–272. American Society of Mechanical Engineers, 2010.

[64] Rolf Findeisen and Frank Allgöwer. An introduction to nonlinear model predictive control. In *21st Benelux Meeting on Systems and Control*, volume 11, 2002.

[65] J Anthony Rossiter. *Model-based predictive control: a practical approach*. CRC press, 2013.

[66] Raphael Cagienard, Pascal Grieder, Eric C Kerrigan, and Manfred Morari. Move blocking strategies in receding horizon control. *Journal of Process Control*, 17(6):563–570, 2007.

About the author

The author playing with two trucks

My name is Rui Filipe Oliveira, I was born in 1991 and lived my whole life in Lisbon, Portugal. I studied Electrical and Computer Engineering at Técnico Lisboa, and was accepted in a Dual Master program on Systems, Control and Robotics, spending one year at my home university and the following year at KTH Royal Institute of Technology, where I did this Master Thesis. You can reach me at `rfoli@kth.se`