ÉCOLE CENTRALE DE NANTES

UNIVERSITÀ DEGLI STUDI DI GENOVA

MASTER ERASMUS MUNDUS

EMARO+ "European Master on Advanced RObotics"

2022 / 2023

Master Thesis Report

Presented by

Simone Contorno

21th August 2023

Nonlinear Model Predictive Control for Self-Driving Vehicles

Supervisors:	Olivier KERMORGANT	Associate Professor (ECN)
	Elwan HÉRY	Associate Professor (ECN)
(EMARO+)	Carmine RECCHIUTO	Assistant Professor (UniGe)

Laboratory: Laboratoire des Sciences du Numérique de Nantes LS2N

Abstract

Self-driving vehicles are one of the hottest topics in recent years. Their application on the road requires being able to move inside a dynamic environment, avoiding both static and dynamic obstacles (e.g. pedestrians) while reaching the goal position. If an obstacle is static, it can be simply avoided in advance. On the other hand, if the obstacle is dynamic, it is not simple to predict its behaviour. Then, a prediction model is required to continuously update a feasible safe trajectory in real-time.
Environment dynamicity implies nonlinear systems. Finding a suitable solution to nonlinear optimization problems within real-time requirements is a current challenging research topic.
This document gives an initial review of Self-driving vehicles, Quadratic Programming and Model Predictive Control. Then, it reports the carried out work during my internship at the LS2N (Laboratoire des Sciences du Numérique de Nantes) for the Master Thesis in Robotics Engineering, where I developed a C++ Nonlinear Model Predictive Control in ROS 2 based on the Sequential Quadratic Programming method by recursively calling the ProxQP solver. In the end, the obtained results are shown, for different models with fixed and dynamic obstacles, highlighting good achievements.

Keywords: Self-driving vehicles — Quadratic Programming — Model Predictive Control — Nonlinear optimization problems

Notations

J()	Objective function
L()	Lagrangian function
h()	Equality constraints function
g()	Inequality constraints function
f()	State function
$\mu()$	Control input parameterization function
$\mathbf{z}$	Decision variables vector
$\mathbf{c}$	Linear objective function coefficients vector
$\mathbf{b}$	Equality constraints vector
$\mathbf{d}$	Inequality constraints vector
$\boldsymbol{\lambda}$	Equality constraints Lagrange multipliers vector
$\boldsymbol{\mu}$	Inequality constraints Lagrange multipliers vector
$\mathbf{w}$	Slack variables vector
$\mathbf{x}_k$	Vehicle state vector at *k-th* time instant
$\mathbf{u}_k$	Vehicle control input vector at *k-th* time instant
$\mathbf{s}_k$	State initial guess vector at *k-th* time instant
$\mathbf{d}_x$	State inequality constraints vector
$\mathbf{d}_u$	Control input inequality constraints vector
$\mathbf{w}_x$	State slack variables vector
$\mathbf{w}_u$	Control input slack variables vector
$\mathbf{U}$	Control input parameterization vector
$\mathbf{H}$	Hessian matrix
$\mathbf{E}$	Equality constraints coefficients matrix
$\mathbf{C}$	Inequality constraints coefficients matrix
$\mathbf{K}$	Karush–Kuhn–Tucker matrix
$\mathbf{x}$	Vehicle states matrix
$\mathbf{u}$	Vehicle control inputs matrix
$\mathbf{s}$	States initial guesses matrix
$\mathbf{A}$	State coefficients matrix
$\mathbf{B}$	Control input coefficients matrix
$\mathbf{C}_x$	State inequality constraints coefficients matrix
$\mathbf{C}_u$	Control input inequality constraints coefficients matrix
$\mathbf{S}$	Final state weighting matrix
$\mathbf{Q}$	States weighting matrix
$\mathbf{R}$	Control inputs weighting matrix
$\mathbf{W}_{w_x}$	State constraints relaxing weighting matrix
$\mathbf{W}_{w_u}$	Control input constraints relaxing weighting matrix
n_z	Quadratic Programming problem dimension
n_e	Equality constraints dimension
n_i	Inequality constraints dimension
n_h	Hierarchy levels
n_p	Prediction horizon dimension
n_c	Control horizon dimension
n	Vehicle state vector dimension
m	Vehicle control input vector dimension
Δt	Step size
T	Prediction horizon time

Abbreviations

AMCL	Adaptive Monte Carlo Localization
CPU	Central Processing Unit
EKF	Extended Kalman Filter
GPS	Global Positioning System
ICS	Inevitable Collision States
IMUs	Inertial Measurement Units
KKT	Karush-Kuhn-Tucker
LiDAR	Light Detection And Ranging
LMPC	Linear Model Predictive Control
LS2N	Laboratoire des Sciences du numérique de Nantes
MPC	Model Predictive Control
MSBPC	Multi-Sensor-Based Predictive Control
NMPC	Nonlinear Model Predictive Control
OPDF	Occupancy Probability Density Function
QP	Quadratic Programming
RAM	Random Access Memory
RHC	Receding Horizon Control
ROS	Robot Operating System
SLAM	Simultaneous Localization And Mapping
SQP	Sequential Quadratic Programming
TEB	Timed-Elastic-Band
URDF	Unified Robot Description Format

List of Figures

List of Tables

List of Algorithms

Contents

Big Picture

Self-driving vehicles are an important technology with the potential of increasing both efficiency and safety of road navigation. They are endowed with a combination of proprioceptive, e.g. *Inertial Measurement Units (IMUs)*, and exteroceptive, e.g. *Global Positioning System (GPS)* and *Light Detection And Ranging (LiDAR)*, sensors to simultaneously localize the own position and build a map. The collected sensor data are elaborated by the *perception system*, resulting in an internal representation of the environment; this is sent to the *decision-making system*, which produces a sequence of efforts (trajectory) as vehicle inputs. In Fig. 1 it is shown how the Vehicle sends its state and the sensor data to the Perception system which elaborates an Internal representation of the environment; this is sent to the Decision-making system, where in sequence: the Route planner generates a Route, the Path planner generates a set of Paths, the Behaviour selector chooses a Path and a Goal according to the driving behaviour and avoiding collisions within the decision time horizon, the Motion planner computes a trajectory, the Obstacle avoider corrects the trajectory to avoid collisions, and the Controller sends Efforts commands to the Vehicle [6].
Obstacle avoidance is one of the most important tasks during autonomous navigation; it requires 3 main levels [7]: perception, e.g. *Simultaneous Localization and Mapping (SLAM)* [8], planning, e.g. *Timed-Elastic-Band (TEB)* [9], and control, e.g. *Model Predictive Control (MPC)* [10]. The required trajectory to reach the goal position must not only be feasible but also safe for the driver, the passengers, and the pedestrians [11]. Therefore, the control systems should be designed in order to avoid, or at least minimize as much as possible, the risk of hurt: the faster the optimization problem is solved, the faster a safe trajectory can be predicted (Fig. 2).
The present document is structured as follows:

- Chapter 1 introduces the state of the problem;
- Chapter 2 recalls the working principle of Quadratic Programming, Sequential Quadratic Programming, and Model Predictive Control;
- Chapter 3 explains the developed Nonlinear Model Predictive Control and provides the configurations for the different vehicles (virtuals and reals) used in the experiments;
- Chapter 4 details the experiments set up and reports the results obtained for the different models (unicycle and bicycle) used under different scenarios (fixed and dynamic obstacle);
- Chapter 5 gives personal suggestions for possible improvements.

The carried out work provides a new Nonlinear Model Predictive Control [12] written in C++ and running in ROS 2 (Robot Operating System). The program highlighted good real-time performances (Tab. 4.1 and 4.2), allowing real-vehicle applications (Sec. 3.8). Moreover, this has been written to be versatile and then usable with different vehicle models (Chap. 3).

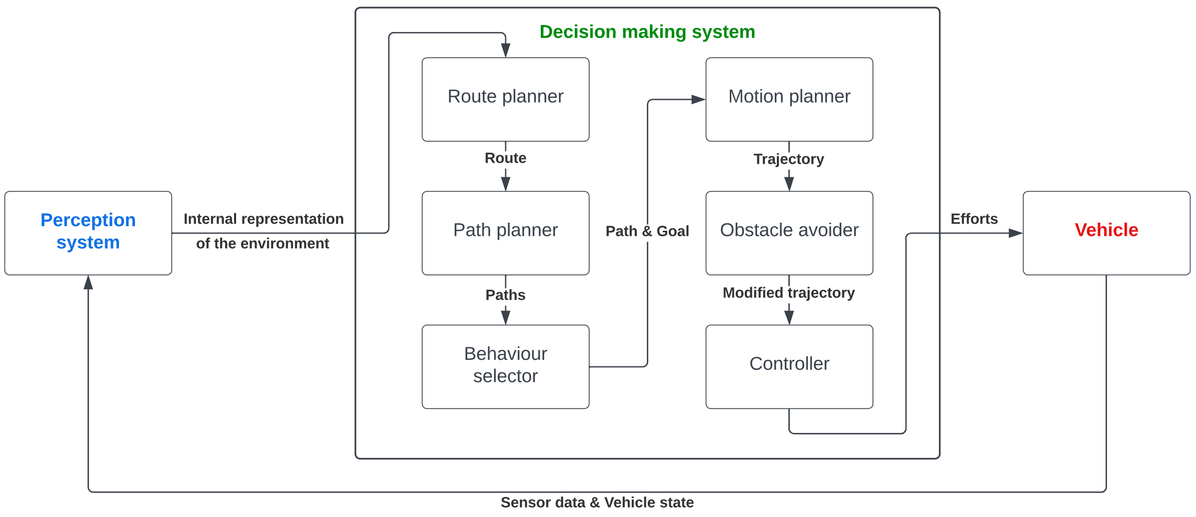


Figure 1: Autonomous vehicle architecture.

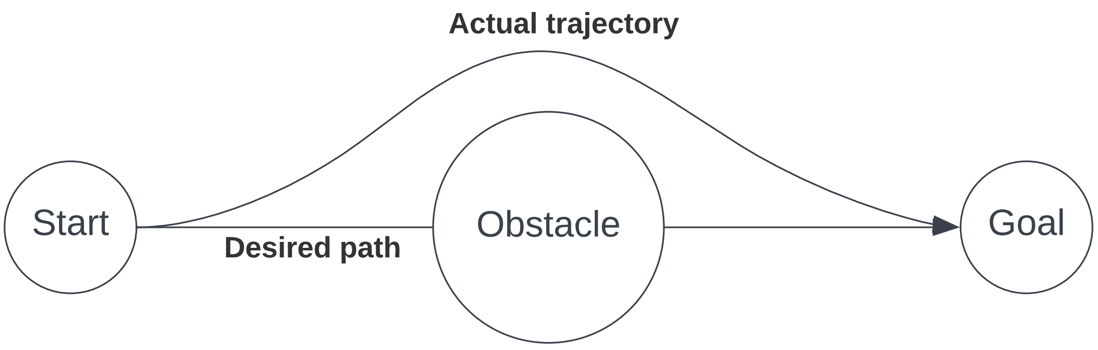


Figure 2: Feasible and safe trajectory estimated by the autonomous vehicle.

Chapter 1

Introduction

1.1 Problem statement

In a dynamic environment, *obstacle/collision avoidance* is a challenging task since the term includes not only fixed obstacles but also dynamic ones, like pedestrians. A hard situation can be well-represented at road intersections where a high number of collisions is reported. For improving the level of safety in these types of situations a behaviour analysis of vehicles, drivers, and pedestrians can be done (Fig. 1.1). In particular, predicting pedestrian behaviour is not simple, since a lot of factors can influence their crossing choice and motion pattern [13].
A possibility to address the obstacle avoidance task is using a Multi-Sensor-Based Predictive Control (MSBPC) [14] which tries to minimize the difference between current sensor feature values and the desired ones, under multi-sensor feature constraints; this approach has the advantage of not needing of any path planning, which requires to have knowledge about free and occupied space of the whole environment and it is strictly dependent on the localization performance. On the other hand, the main disadvantage is the possibility to fall into local minima, then a feasible safe trajectory must be planned.
The Model Predictive Control (MPC) is a control strategy that minimizes a given objective function under certain equality and inequality constraints, in order to generate a sequence of inputs over a finite time horizon (trajectory).
Due to the dynamicity of the environment, the system will be nonlinear and the corresponding MPC will be a Nonlinear MPC (NMPC) able to handle both nonlinear objective function and constraints. Solving nonlinear optimization problems requires higher computational costs, which we want to minimize as much as possible. This represents a current challenging research topic in the NMPC field.

1.2 World model prediction

As already anticipated, obstacle avoidance is the highest priority task that ensures safe navigation. For this reason, the car must predict the collisions sufficiently in advance, in order to compute the optimal trajectory and avoid them.
If the prediction time is not sufficiently large and/or the car does not take a decision in time, it will enter in the so-called *Inevitable Collision States (ICS)* ad the collision will occur.
Then, *motion safety* requires 3 rules:

- decision time must be upper-bounded,
- time horizon must be lower-bounded, and
- reasoning about the future, globally considering the obstacles, is required.

The last one is due to the fact that "the set of ICS generated by a set of obstacles is not the union of the set of ICS generated by each obstacle independently" [2], i.e., considering each obstacle independently does not ensure collision avoidance for all of them (Fig. 1.2).
To represent the future world model there are 3 approaches: *deterministic*, *conservative*, and *probabilistic*.

Deterministic approach The world model is predicted by the nominal future motion of each obstacle, but if it is fixed, then motion safety is ensured as long as the environment does not change (not realistic).

Conservative approach Based on the reachable set of each obstacle, all possible future motions are considered. This guarantees motion safety, but in the long run (infinite time horizon) every state for the vehicle will become an ICS (too pessimistic).

Probabilistic approach The obstacle position at each time instant is represented by an *Occupancy Probability Density Function (OPDF)*. In this way, the uncertainty about the future motion of the obstacles is well represented, but over time (infinite time horizon) the OPDF will diffuse, i.e., the obstacles will be nowhere (too optimistic).

Considering only the braking trajectory it is possible to reason over a finite time horizon. At this point the conservative approach can be adopted, ensuring the safety motion.
In this case, if the vehicle enters in a *Braking ICS* it will have the time to stop before the collision, and the moving obstacle (which is assumed to be endowed of reasoning, as happens in most cases) will have the time to avoid it (*Passive Friendly Safety* [15]).

1.3 Collision avoidance

In order to avoid collisions while performing trajectory tracking, sensor-based approaches can be adopted to detect in real-time the presence of obstacles. For example, the use of a LiDAR sensor provides detailed 3D point cloud data, enabling accurate obstacle detection and localization.
Once this information is acquired, an optimal trajectory can be computed in order to follow the reference one maintaining a minimum distance from the detected obstacles. There are various techniques aiming at this purpose, e.g. *Dynamic Window Approach* [16], but the needing of having a prediction by considering future vehicle and obstacle states for safer trajectories leads to the use of a *Model Predictive Control*.
An important fact that must be taken into account is that implementing collision avoidance requires the managing of a consistent amount of data, provided by different sensors, which are commonly combined through the use of sensor fusion with *Extended Kalman Filter (EKF)* methods [17]. This approach is commonly used to estimate the dynamic obstacle trajectory [18]. In the case of pedestrians, there are a lot of techniques to predict their behaviour [19] and their trajectory [20], but for this work, the obstacle (fixed and dynamic) was simulated and its trajectory was given without needing estimation (Sec. 3.4).
Moreover, notice that introducing constraints for collision avoidance in optimization algorithms increases considerably the computational load required to find an optimal solution.

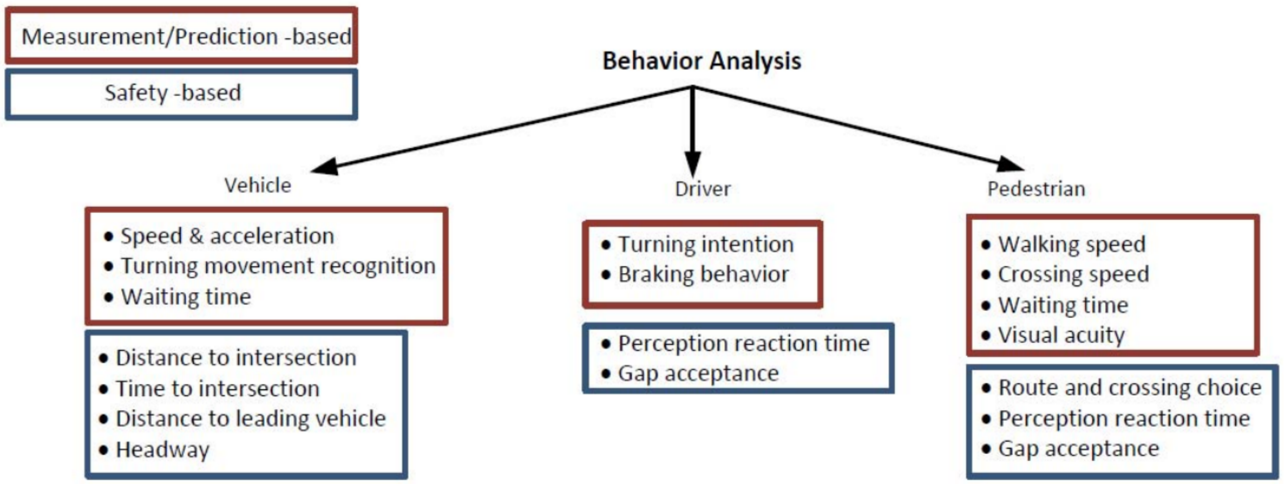


Figure 1.1: A typical structure of behaviour analysis at intersections [1].

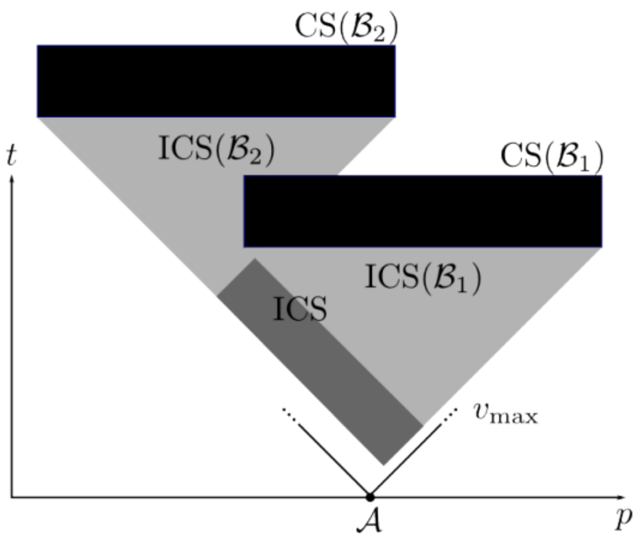


Figure 1.2: State $\times$ time space for the compactor scenario: the set of ICS generated by a set of obstacles ICS(B_1) and ICS(B_2) [2].

Chapter 2

State of the art

2.1 Quadratic Programming

Quadratic Programming (QP) is a method to solve linear optimization problems finding the optimal values of the *decision variables* that minimize a quadratic *objective function* (convex) while satisfying linear constraints. QP is used for trajectory optimization, thanks to its capability to handle complex systems with both equality and inequality constraints. In particular, the *Sequential Quadratic Programming (SQP)* (Sec. 2.2) is used to solve nonlinear optimization problems by solving quadratic sub-problems [21].

2.1.1 Formulation

A Quadratic Program can be formulated as follow:

$$\underset{\mathbf{z}}{argmin}\ J(\mathbf{z}) = \frac{1}{2}\,\mathbf{z}^T\mathbf{H}\mathbf{z} + \mathbf{c}^T\mathbf{z}$$
$$s.t.:\begin{cases}\mathbf{E}\mathbf{z} = \mathbf{b}\\ \mathbf{C}\mathbf{z} \leq \mathbf{d}\end{cases} \tag{2.1}$$

where $\mathbf{z} \in \mathbb{R}^{n_z}$ is the *decision variables vector*, $\mathbf{H} \in \mathbb{R}^{n_z \times n_z}$ is the symmetric *Hessian matrix*, and $\mathbf{c} \in \mathbb{R}^{n_z}$ is the *linear objective function coefficients vector*, with n_z the *problem dimension*; $\mathbf{b} \in \mathbb{R}^{n_e}$ and $\mathbf{E} \in \mathbb{R}^{n_e \times n_z}$ are respectively the *equality constraints vector* and the *equality constraints coefficients matrix*, with n_e the number of equality constraints; $\mathbf{d} \in \mathbb{R}^{n_i}$ and $\mathbf{C} \in \mathbb{R}^{n_i \times n_z}$ are respectively the *inequality constraints vector* and the *inequality constraints coefficients matrix*, with n_i the number of inequality constraints.
If $\mathbf{H}$ is positive definite ($\mathbf{z}^T\mathbf{H}\mathbf{z} > 0\ \forall \mathbf{z} \in \mathbb{R}^{n_z}$), (2.1) is a *Strictly convex QP* and the objective function $J(\mathbf{z})$ has a unique global minima. If $\mathbf{H}$ is positive semi-definite ($\mathbf{z}^T\mathbf{H}\mathbf{z} \geq 0\ \forall \mathbf{z} \in \mathbb{R}^{n_z}$), (2.1) is a *convex QP* and $J(\mathbf{z})$ can have more than one global minima solution. If $\mathbf{H}$ is indefinite (neither positive semi-definite nor negative semi-definite), (2.1) is a *Non-convex QP* and in this case $J(\mathbf{z})$ can have several stationary points and local minima.

2.1.2 Equality-constrained QP

If the QP has only equality constraints, the optimization problem (2.1) is only subject to

$$\mathbf{E}\mathbf{z} = \mathbf{b} \tag{2.2}$$

Denoting h($\mathbf{z}$) = $\mathbf{E}\mathbf{z}$ - $\mathbf{b}$ and $\boldsymbol{\lambda}$ the *Lagrange multipliers* vector, in this case, the problem can be solved using the *Lagrangian* function

$$L(\mathbf{z}, \boldsymbol{\lambda}) = J(\mathbf{z}) + \boldsymbol{\lambda}^T h(\mathbf{z}) \tag{2.3}$$

which tries to find the global extremum (in this case, the minimum) of the objective function. The solution is a *saddle point* of the Lagrangian, where all the partial derivatives (concerning both $\mathbf{z}$ and $\boldsymbol{\lambda}$) are 0.
The problem (2.1) can now be written in matrix form as

$$\begin{bmatrix} \mathbf{H} & \mathbf{E}^T \\ \mathbf{E} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{z} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \mathbf{b} \end{bmatrix} \tag{2.4}$$

where the first matrix on the left, which here will be denoted as $\mathbf{K}$, takes the name of *Karush–Kuhn–Tucker (KKT) matrix.* If $\mathbf{H}$ is positive definite and $\mathbf{E}$ is full rank (no constraint redundancies), $\mathbf{K}$ is non-singular and there is a unique vector pair ($\boldsymbol{z}^*$, $\boldsymbol{\lambda}^*$) which is the solution of the problem (2.1) with only equality constraints ([22] section 16.1).

2.1.3 Inequality-constrained QP

If also inequality constraints are present, the *Lagrangian* (2.3) must be generalized using *KKT* conditions, which are known as the *First-Order Necessary Conditions*:

$$\begin{cases} \bigtriangledown_x L(\mathbf{z}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = 0 \\ h(\mathbf{z}) = \mathbf{E}\mathbf{z} - \mathbf{b} = 0 \\ g(\mathbf{z}) = \mathbf{C}\mathbf{z} - \mathbf{d} \leq 0 \\ \boldsymbol{\mu}^* \geq 0 \\ \boldsymbol{\mu}^* g(\mathbf{z}) = 0 \end{cases} \tag{2.5}$$

where $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are the *KKT multipliers* vectors.
The 1st condition takes the name of *Stationarity* and imposes that the problem solution is a saddle point of the *Lagrangian*, i.e., all the partial derivatives are equal to 0. The 2nd and 3rd ones are the *Primal feasibility* which impose that both equality and inequality constraints are satisfied. The 4th condition called *Dual feasibility*, tells that the KKT multipliers associated with the inequality constraints must be greater or equal to 0. The last condition is the so-called *Complementary slackness* which describes the relationship between the inequality constraints and the associated KKT multipliers: if an inequality constraint is active ($g(z_k) = 0$) the corresponding KKT multiplier is positive; conversely, if the inequality constraint is inactive ($g(z_k) \leq 0$) the corresponding KKT multiplier is equal to 0.
A QP with both equality and inequality constraints first solves the problem without inequalities, and then those that have been violated are turned into equalities for the next iteration; if a KKT multiplier of an active inequality constraint is less or equal to zero, this is deactivated.

2.1.4 Hierarchical QP

For handling conflicts between equality and inequality constraints, a hierarchy can be established, but this leads to potentially violating lower priority constraints. Then, the aim is to minimize this violation. The *Hierarchical QP* tries to find a solution to (2.1) also minimizing the so-called *slack variables* $\mathbf{w}$, which tells us how much the inequality constraints are violated. The problem is now formulated as follows:

$$\begin{aligned} \mathbf{z}_k, \mathbf{w}_k = & \underset{\mathbf{z},\mathbf{w}}{argmin} \ ||\mathbf{E}_k\mathbf{z} - \mathbf{b}_k||^2 + ||\mathbf{w}||^2 \\ & s.t. : \mathbf{C}_k\mathbf{z} - \mathbf{w} \leq \mathbf{d}_k \end{aligned} \tag{2.6}$$

where $k \in [0, n_h]$ represents the *k-th* hierarchy level.
The inequality constraints that must be activated will be changed into equality constraints for the next hierarchical level: $\mathbf{C}_k\mathbf{z} = \mathbf{d}_k + \mathbf{w}$.

2.2 Sequential Quadratic Programming

Sequential Quadratic Programming (SQP) is a method used to solve nonlinear optimization problems. The idea is to iteratively solve a QP sub-problem and update the decision variables with the obtained results until a convergence test is satisfied [22].

2.2.1 QP sub-problem

The QP sub-problem is constructed starting from (2.1) and performing a local approximation of the nonlinear function [23]. In particular, the objective function is approximated to the second order, while the constraints are to the first one. The resulting problem is:

$$\begin{aligned}
&\underset{\mathbf{z}-\mathbf{z}_k}{argmin}\ J(\mathbf{z}) \approx J(\mathbf{z}_k) + \nabla J(\mathbf{z}_k)\ (\mathbf{z}-\mathbf{z}_k) + \frac{1}{2}\ (\mathbf{z}-\mathbf{z}_k)^T \mathbf{H} J(\mathbf{z}_k)\ (\mathbf{z}-\mathbf{z}_k) \\
&s.t.: \begin{cases} h(\mathbf{z}) \approx h(\mathbf{z}_k) + \nabla h(\mathbf{z}_k)\ (\mathbf{z}-\mathbf{z}_k) = 0 \\ g(\mathbf{z}) \approx g(\mathbf{z}_k) + \nabla g(\mathbf{z}_k)\ (\mathbf{z}-\mathbf{z}_k) \leq 0 \end{cases}
\end{aligned} \tag{2.7}$$

where $h(\mathbf{z}) = \mathbf{E}\mathbf{z} - \mathbf{b}$ and $g(\mathbf{z}) = \mathbf{C}\mathbf{z} - \mathbf{d}$.

2.2.2 ProxQP

A new QP approach has been proposed in [3], where the KKT conditions are used to take into account also inequality constraints, defining some *stopping criterion* subject to an accuracy parameter that defines the solver precision. The proposed solver, called *ProxQP*, introduces new penalty terms to the *Lagrangian* function.
The goal is to find the triplet, composed of the primal variable (input) and the dual ones (KKT multipliers), that satisfies the stopping criterion.
The algorithm solves recursively until the stopping criterion is not satisfied, proximal sub-problems and check if the (primal) infeasibility (i.e., the maximum value between the equality and inequality constraints violation) is greater than the accuracy parameter; in this case, the penalty terms on the constraints are increased, otherwise they remain unchanged and the dual variables are updated.
By the results [3], the ProxQP algorithm highlighted the fastest performance among different QP solvers, under different problem scenarios. In Fig. 2.1 is shown how the solving time for the ProxQP solver remains of the same order of magnitude while the problem dimension grows.

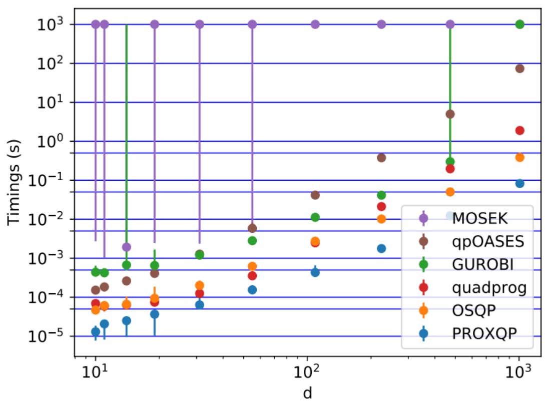


Figure 2.1: Solving times for sparse equality and inequality constrained QPs. For each set of executions, the median is shown with a dot [3]. On the abscissa is reported the problem dimension.

2.3 Model Predictive Control

Using QP there is no prediction and local minima cannot be avoided.
To address optimization problems, *direct methods* allow to obtain a time-discrete *state prediction equation*

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) \ , \ \forall k \in [0, n_p - 1] \tag{2.8}$$

with $\mathbf{x} \in \mathbb{R}^{n_p \times n}$ the *vehicle states matrix* and $\mathbf{u} \in \mathbb{R}^{n_p \times m}$ the *vehicle control inputs matrix*, and starting from an initial guess they perform a local search to find a minimum (which could be local or global). This makes them the most suitable choice for real-time applications; in fact, a "local" search leads to faster performances than a "global" one (e.g. *indirect methods*).
The *Receding Horizon Control (RHC)* is a control strategy which consists in solving at each iteration an open-loop optimal control problem over a finite time horizon, obtaining an optimized control trajectory that is applied until a new one is available at the next iteration. Since each optimized control trajectory is based on the most recent state information, the resulting control is a closed-loop.
The *Model Predictive Control (MPC)* is an open-loop optimization problem (direct) method based on the RHC which aims to find the best control inputs sequence $\mathbf{u}$, over a (finite) *time prediction horizon*. For each predicted trajectory only the first control input $\mathbf{u}_0$ is sent to the vehicle. Once new measurements are available, a new trajectory is predicted (Fig. 2.2).
This method allows for solving online optimization problems, where at each iteration the initial state is the current one of the controlled system.
More in detail, direct methods can use *sequential* (e.g. *Single Shooting*) or *simultaneous* (e.g. *Multiple Shooting*) approaches [24] [25].

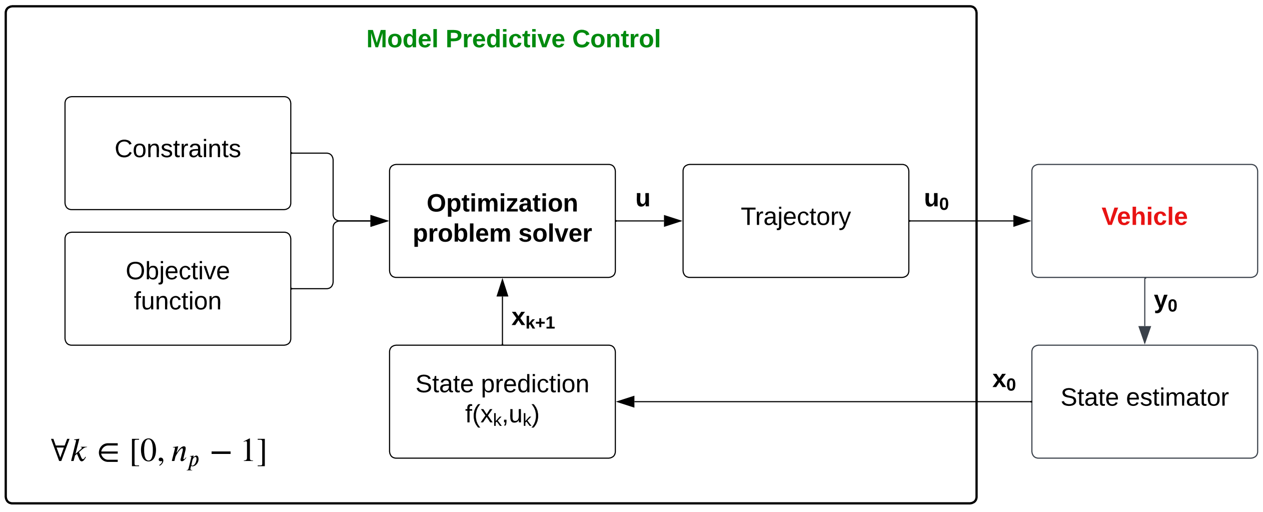


Figure 2.2: Model Predictive Control architecture for a self-driving vehicle.

2.3.1 Direct Single Shooting approach

The Direct Single Shooting approach minimizes the objective function only with respect to the control inputs $\mathbf{u}$ and, as a consequence, only one initial guess

$$\boldsymbol{u}_k \ \forall k \in [0, n_p - 1]$$

is required at each step. The equation (2.8) is solved sequentially (Fig. 2.3a). These characteristics lead to a smaller but dense optimization problem in which computational cost increases at each time step (Fig. 2.3a):

$$\begin{aligned}
&\mathbf{x}_1 = f(\mathbf{x}_0, \mathbf{u}_0)\\
&\mathbf{x}_2 = f(\mathbf{x}_1, \mathbf{u}_1) = f(f(\mathbf{x}_0, \mathbf{u}_0), \mathbf{u}_1)\\
&...\\
&\mathbf{x}_{n_p} = f(\mathbf{x}_{n_{p-1}}, \mathbf{u}_{n_{p-1}}) = f(f(...f(\mathbf{x}_0, \mathbf{u}_0)...), \mathbf{u}_{n_{p-1}})
\end{aligned} \tag{2.9}$$

As can be seen, the state variables are dependent by the first state $\boldsymbol{x}_0$, then in the final equation all the intermediate states $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_{n_{p-1}}$ will be disappeared, while all the control inputs $\boldsymbol{u}_0, \boldsymbol{u}_1, ..., \boldsymbol{u}_{n_{p-1}}$ will be present.

2.3.2 Direct Multiple Shooting approach

The Direct Multiple Shooting approach minimizes the objective function with respect to both the system states $\mathbf{x}$ and the control inputs $\mathbf{u}$; as consequence, two initial guesses

$$\boldsymbol{u}_k \ \forall k \in [0, n_p - 1] \ \ \& \ \ \boldsymbol{s}_k \ \forall k \in [0, n_p - 1]$$

are required at each step ($\mathbf{s} \in \mathbb{R}^{n_p \times n}$ is the *states initial guesses matrix*). In this case, the optimization problem is bigger but sparse, in fact, n_p equations (2.8) are solved simultaneously (Fig. 2.3b):

$$\begin{aligned}
&\mathbf{x}_1 = f(\mathbf{s}_0, \mathbf{u}_0)\\
&\mathbf{x}_2 = f(\mathbf{s}_1, \mathbf{u}_1)\\
&...\\
&\mathbf{x}_n = f(\mathbf{s}_{n_{p-1}}, \mathbf{u}_{n_{p-1}})
\end{aligned} \tag{2.10}$$

The presence of both $\boldsymbol{s}_k$ and $\boldsymbol{x}_k$ for the state at each time step, except for the first one where they coincide, leads to a discontinuity which is solved by imposing the equality constraint:

$$\mathbf{s}_k = \mathbf{x}_k \ , \ \forall k \in [0, n_{p-1}] \tag{2.11}$$

Despite the bigger size (more decision variables) than the Single shooting approach, the sparsity guarantees faster performances, and this makes the Multiple shooting approach more suitable for online optimization.

2.3.3 Linear Model Predictive Control

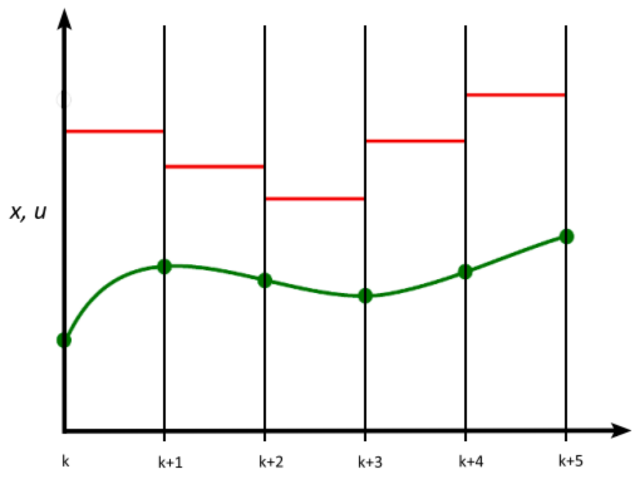
If the time-discrete system is linear then the prediction state equation is:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \ , \ \forall k \in [0, n_p - 1] \tag{2.12}$$

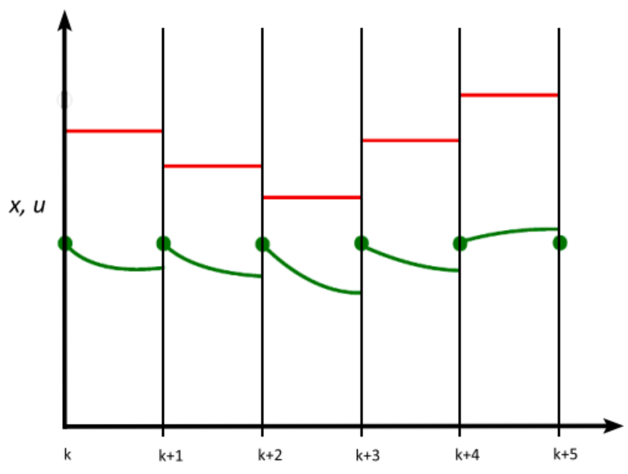
with $\mathbf{A} \in \mathbb{R}^{n \times n}$ the *state coefficients matrix*, and $\mathbf{B} \in \mathbb{R}^{n \times m}$ the *control input coefficients matrix.*
At each time step, the resulting Linear MPC (LMPC) tries to find the control input sequence $\mathbf{u}$ that minimizes the linear objective function $J(\mathbf{x}, \mathbf{u})$ while satisfying the linear constraints over the time prediction horizon:

$$\begin{aligned}
&\mathbf{u} = \underset{\mathbf{x},\mathbf{u}}{argmin} \ J(\mathbf{x}, \mathbf{u})\\
&s.t. \ \forall k \in [0, n_p - 1] : \begin{cases} \mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \\ \mathbf{C}_x\mathbf{x}_k \le \mathbf{d}_x \\ \mathbf{C}_u\mathbf{u}_k \le \mathbf{d}_u \end{cases}
\end{aligned} \tag{2.13}$$

where $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$ are the linear equality constraints, while $\mathbf{C}_x\mathbf{x}_k \le \mathbf{d}_x$ and $\mathbf{C}_u\mathbf{u}_k \le \mathbf{d}_u$ are the linear inequality constraints respectively for the state and the control input, with $\mathbf{C}_x \in \mathbb{R}^{n \times n}$, $\mathbf{d}_x \in \mathbb{R}^n$, $\mathbf{C}_u \in \mathbb{R}^{m \times m}$, and $\mathbf{d}_u \in \mathbb{R}^m$.



(a) Direct Single Shooting approach. Since each state is dependent on the first one, the state evolution is continuous.



(b) Direct Multiple Shooting approach. Since each state is dependent on an initial guess which is not equal to the previous state, the state evolution is discontinuous.

Figure 2.3: Graphical comparison between (Direct) Single and Multiple shooting approaches [4]. The state evolution $\mathbf{x}$ is represented by the green line, while the control input evolution $\mathbf{u}$ is represented by the red one (piece-wise constant).

2.3.4 Nonlinear Model Predictive Control

Self-driving vehicles (e.g. cars) navigate in a dynamic environment, i.e., subject to constant changes due to new events. Therefore, it is necessary to consider nonlinear systems. For this reason, we need to use a *Nonlinear Model Predictive Control (NMPC).*
Using NMPC we can handle nonlinear objective functions subject to nonlinear equality and inequality constraints.
Therefore, the algorithm must solve nonlinear optimal control problems at a high frequency which requires a high online computational load [24]; as consequence, guaranteeing a global (or at least sufficiently good) solution within the real-time requirements is not always feasible.
To ensure the optimization problem feasibility (i.e., a unique solution exists), the KKT conditions (2.5) must be satisfied.
Since nonlinear optimization problems are generally non-convex, the convergence of a given optimization algorithm largely depends on the initial guess(es) provided for the solution. In this sense, the Direct Multiple Shooting approach allows handling an initial guess also for the state trajectory (section 2.3.2).
The NMPC can be formulated as (2.13) with the difference that the objective function and the constraints are nonlinear; the prediction state equation looks like (2.8).

Stability

Due to the nonlinearity, ensuring stability for NMPC, i.e., guaranteeing that the nonlinear constraints on both the states and the control inputs are always satisfied, is harder than for LMPC. However, there are several ways to ensure stability [26]. In general [25]:

- the *parameterization of the control input* $\mathbf{u}(t) = \mu(t, \mathbf{U})$ ($\mathbf{U}$ is the *control input parameterization vector*) must be sufficiently flexible, must not contain unnecessary parameters, and must be implementable (it is generally chosen piece-wise constant,

$$\mathbf{u}(t) = \mu(t, \mathbf{U}) = U_k \ , t_k < t < t_{k+1} \ , \ \forall k \in [0, n_p - 1]);$$

- a larger *prediction horizon* improves the stability, but increases the computational costs and the resulting prediction is less accurate;
- feasible *terminal set of constraints* $\mathbf{x}_{n_p} \in X$, *terminal equality constraints* $\mathbf{x}_{n_p} = \mathbf{x}^*_{n_p}$, and *terminal cost* $S(x_{n_p})$ (weighted by the positive symmetric matrix $\mathbf{S} \in \mathbb{R}^{n\times n}$) can be combined to ensure the system convergence to a desired final state at the end of the prediction horizon.

Moreover, the NMPC can be tuned by two positive symmetric weighting matrices $\mathbf{Q}$ and $\mathbf{R}$:

$$J(\mathbf{x}, \mathbf{u}) = \ ||\mathbf{x}_{n_p} - \mathbf{x}^*_{n_p}||^2_{\mathbf{S}} + \sum_{k=0}^{k=n_p-1} ||\mathbf{x}_k - \mathbf{x}^*_k||^2_{\mathbf{Q}} + ||\mathbf{u}_k||^2_{\mathbf{R}} \tag{2.14}$$

where $||\mathbf{x}_{n_p} - \mathbf{x}^*_{n_p}||^2_{\mathbf{S}}$ is the terminal cost. The $\mathbf{Q} \in \mathbb{R}^{n\times n}$ matrix refers to the vehicle state, while the $\mathbf{R} \in \mathbb{R}^{m\times m}$ matrix refers to the vehicle control input.

Note For a general vector $\mathbf{v}$ and a general weighting matrix $\mathbf{M}$: $||\mathbf{v}||^2_{\mathbf{M}} = \mathbf{v}^T\mathbf{M}\mathbf{v}$.

Using 2.14, a general NMPC formulation follows:

$$\begin{aligned} \mathbf{u} = \ & \underset{\mathbf{u},\mathbf{x}}{argmin} \ J(\mathbf{x}, \mathbf{u}) \\ s.t.\ \forall k \in [0, n_p - 1] : & \begin{cases} \mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) \\ \mathbf{x}_{k+1} \in X \\ \mathbf{u}_k \in U \end{cases} \end{aligned} \tag{2.15}$$

where X and U are valid domains respectively for the state and the control input.

Slack variables To ensure NMPC feasibility as far as possible, the inequality constraints can be relaxed by introducing the *slack variables* $\boldsymbol{w}$, which can be applied for both the state ($\mathbf{w}_x$) and the control input ($\mathbf{w}_u$). These terms are added to the objective function and are weighted by two positive matrices $\mathbf{W}_{w_x}, \mathbf{W}_{w_u}$. In this case, a possible formulation of the NMPC is:

$$\begin{aligned} \mathbf{u} = \ & \underset{\mathbf{x},\mathbf{u},\mathbf{w}}{argmin} \ J(\mathbf{x}, \mathbf{u}) + ||\mathbf{W}_{w_x}\mathbf{w}_x||_1 + ||\mathbf{W}_{w_u}\mathbf{w}_u||_1 \\ s.t.\ \forall k \in [0, n_p - 1] : & \begin{cases} \mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) \\ \mathbf{C}_x\mathbf{x}_k - \mathbf{w}_x \le \mathbf{d}_x \\ \mathbf{C}_u\mathbf{u}_k - \mathbf{w}_u \le \mathbf{d}_u \end{cases} \end{aligned} \tag{2.16}$$

where $||\cdot||_1$ is the l_1 norm which consists of the sum of all the vector elements.
Inequality constraints can be converted into equalities when necessary, minimizing as much as possible their violation $\mathbf{w}$. Slack variables can be applied also to equality constraints, but this increases the computational costs and can lead to infeasibility problems.
Since the optimization is repeated at each loop iteration, the solver may not always converge, but should generate a feasible solution; slack variables can be used to relax the constraints when possible, slightly changing the problem formulation in order to guarantee feasibility.
In addition, at each optimization run the *warm start* can be adopted: the solution of the last iteration is used for initializing the next one, e.g. the inequality constraints that have been converted into equalities can be maintained, and the initial guesses can be set by the previous run.

Chapter 3

Development and Implementation

3.1 Introduction

During the internship at the LS2N (Laboratoire des Sciences du numérique de Nantes) I developed a NMPC in ROS 2 written in C++ [12]; the Algorithms 1 and 2 show respectively the pseudo-code of the implemented NMPC and the QP solver based on ProxQP.
I started with the implementation of a LMPC for the trajectory tracking task. Then, I adapted it to the nonlinear case using the SQP method (Sec. 2.2) by recursively solving QP sub-problem with the ProxQP solver (Sec. 2.2.2). Finally, I added a constraint to test the performances with the obstacle avoidance activated (Sec. 3.4).
The program has been developed in order to be adapted to various vehicle models, e.g. unicycle (Fig. 3.1a) and bicycle (Fig. 3.1b), by correctly configuring the problem. Moreover, the control horizon can be shorter than the prediction one, since this can save computation load [27].
Notice that in the following problem formulations, the reference model is the bicycle one:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} v\ cos(\theta+\beta) \\ v\ sin(\theta+\beta) \\ v\ sin(\beta)/L \\ w \end{bmatrix} \tag{3.1}$$

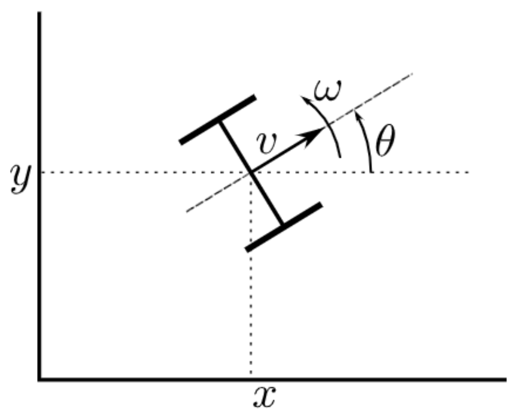
where:

- $[x, y, \theta, \beta]^T$ is the state vector $\mathbf{x}$ composed by the Cartesian coordinates, the orientation and the steering angle (front wheel);
- $[v, w]^T$ is the control vector $\mathbf{u}$ composed by the linear (m/s) and the steering (rad/s) velocities;
- L is the distance between the front and the rear wheels.

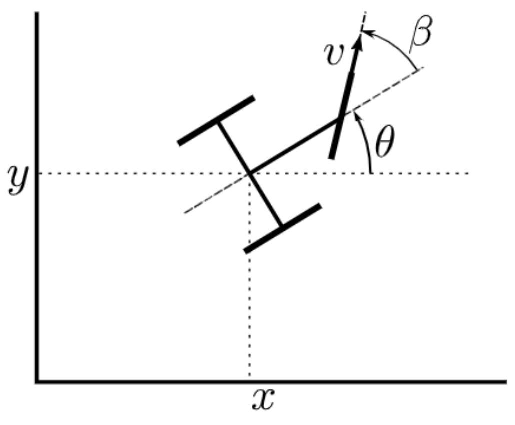
3.1.1 QP solver

The QP solver used to solve the QP sub-problems is ProxQP (Sec. 2.2.2). The problem, similarly to (2.1), is formulated as follow:

$$\begin{aligned} &\underset{\mathbf{z}}{min}\ f(\mathbf{z}) = \frac{1}{2}\ \mathbf{z}^T\mathbf{H}\mathbf{z} + \mathbf{c}^T\mathbf{z} \\ &s.t.: \begin{cases} \mathbf{E}\mathbf{z} = \mathbf{b} \\ \mathbf{d}_{low} \leq \mathbf{C}\mathbf{z} \leq \mathbf{d}_{upp} \end{cases} \end{aligned} \tag{3.2}$$



(a) Unicycle model.



(b) Bicycle model.

Figure 3.1: Vehicle models [5].

Algorithm 1 MyNMPC

```
1: **Slide states and controls by 1 position:**
2: $x^k \leftarrow x^{k+1}\ \forall k \in [0, n_p - 1]$
3: $u^k \leftarrow u^{k+1}\ \forall k \in [0, n_c - 1]$
4:
5: **Initialize initial state and control:**
6: $x^0 \leftarrow \hat{x}$
7: $u^0 \leftarrow u_0$
8:
9: **Local SQP** [22]:
10: **while** ProxQP IS NOT SOLVED **and** $SQP_{iter} < SQP_{max}$ **do**
11:
12:     **Solve the QP sub-problem:**
13:     $[x_{sol}, u_{sol}, w_{sol}] \leftarrow \text{ProxQP}(x, u, u_0, w, x^*, u^*)$
14:
15:     **Update optimal variables:**
16:     $x \leftarrow x + x_{sol}$                ▷ state
17:     $u \leftarrow u + u_{sol}$                ▷ control
18:     $w \leftarrow w + w_{sol}$                ▷ slack variable
19:
20: **end while**
21:
22: **Save the first control input:**
23: $u_0 = u^0$
24:
25: **return** $x, u$
```

Algorithm 2 QP solve function based on ProxQP

```
1: Set the problem (3.2):
2: setH()                                   ▷ Hessian matrix
3: setc($x, u, w, x^*, u^*$)                ▷ linear objective function coefficients vector
4: setE($x, u$)                             ▷ equality constraints coefficients matrix
5: setb($x, u$)                             ▷ equality constraints vector
6: setC($x$)                                ▷ inequality constraints coefficients matrix
7: setd($x, u, u_0, w$)                     ▷ inequality constraints vector
8:
9: Solve it with sparse or dense solver [3]:
10: $[x_{sol}, u_{sol}, w_{sol}] \leftarrow$ solve()
11:
12: Update the decision variables:
13: $x^k \leftarrow x^k_{sol}\ \forall k \in [0, n_p - 1]$     ▷ state
14: $u^k \leftarrow u^k_{sol}\ \forall k \in [0, n_c - 1]$     ▷ control
15: $w^k \leftarrow w^k_{sol}\ \forall k \in [0, n_p - 1]$     ▷ slack variable
16:
17: return $x, u, w$
```

3.2 Linear MPC

Firstly, the control model has been developed as a Linear MPC to track a reference trajectory:

$$\underset{\mathbf{x},\mathbf{u}}{min}\ J(\mathbf{x},\mathbf{u}) = \ ||\mathbf{x}_{n_p} - \mathbf{x}^*_{n_p}||^2_{\mathbf{S}} + \sum_{k=0}^{k=n_p-1} ||\mathbf{x}_k - \mathbf{x}^*_k||^2_{\mathbf{Q}} + \sum_{k=0}^{k=n_c} ||\mathbf{u}_k||^2_{\mathbf{R}}$$
$$s.t.: \begin{cases} (1)\ \mathbf{x}_0 = \hat{\mathbf{x}} \\ (2)\ \mathbf{x}_k - \mathbf{x}_{k+1} + \mathbf{B}_k\mathbf{u}_k = 0\ ,\ \forall k \in [0, n_p - 1] \\ (3)\ \beta_{min} \le \beta_k \le \beta_{max}\ ,\ \forall k \in [0, n_p] \\ (4)\ \mathbf{u}_{prev} + \Delta t\ \dot{\mathbf{u}}_{min} \le \mathbf{u}_0 \le \mathbf{u}_{prev} + \Delta t\ \dot{\mathbf{u}}_{max} \\ (5)\ \mathbf{u}_{min} \le \mathbf{u}_k \le \mathbf{u}_{max}\ ,\ \forall k \in [0, n_c] \\ (6)\ \Delta t\ \dot{\mathbf{u}}_{min} \le \mathbf{u}_{k+1} - \mathbf{u}_k \le \Delta t\ \dot{\mathbf{u}}_{max}\ ,\ \forall k \in [0, n_c - 1] \end{cases} \tag{3.3}$$

where:

- (1) imposes the initial optimal state equal to the current pose of the vehicles ($\hat{\mathbf{x}}$) given by the state estimator;
- (2) imposes the linearized vehicle kinematic model with $\mathbf{B}$ the matrix which describes the kinematics of the vehicle;
- (3) sets the steering angle bounds;
- (4) sets the first optimal control input (the one that will be sent to the vehicle) bounds, given by the last sent ($\mathbf{u}_{prev}$) and the accelerations bounds;
- (5) sets the velocities bounds;
- (6) sets the accelerations bounds.

The objective function $J(\mathbf{x}, \mathbf{u})$ is given by

- the distance between the states and the desired ones along the *prediction horizon* n_p, and
- the control inputs along the *control horizon* n_c.

Note Δt is the sampling time.

3.3 Nonlinear MPC

Secondly, the control model has been converted to a Nonlinear MPC by using the SQP method (Sec. 2.2). Then, the QP problem (3.3) resulted in:

$$\begin{aligned}
\underset{\mathbf{x}-\mathbf{x}_i, \mathbf{u}-\mathbf{u}_i}{min} \; J(\mathbf{x}, \mathbf{u}) &\approx J(\mathbf{x}_i, \mathbf{u}_i) + \nabla J(\mathbf{x}_i, \mathbf{u}_i) \begin{bmatrix} \mathbf{x}-\mathbf{x}_i \\ \mathbf{u}-\mathbf{u}_i \end{bmatrix} \\
&+ \frac{1}{2} \begin{bmatrix} (\mathbf{x}-\mathbf{x}_i)^T & (\mathbf{u}-\mathbf{u}_i)^T \end{bmatrix} \mathbf{H}J(\mathbf{x}_i, \mathbf{u}_i) \begin{bmatrix} \mathbf{x}-\mathbf{x}_i \\ \mathbf{u}-\mathbf{u}_i \end{bmatrix} \; , \; \forall i \in \mathbb{N}_0 \\
s.t. : &\begin{cases}
(1) \; \mathbf{x}^0 - \mathbf{x}_0 = 0 \; , \; \mathbf{x}_0 = \hat{\mathbf{x}} \\
(2) \; \mathbf{A}_k(\mathbf{x}^k - \mathbf{x}_k) - (\mathbf{x}^{k+1} - \mathbf{x}_{k+1}) + \mathbf{B}_k(\mathbf{u}^k - \mathbf{u}_k) = \\
\qquad\qquad - (\mathbf{x}_k - \mathbf{x}_{k+1} + \mathbf{B}_k\mathbf{u}_k) \; , \; \forall k \in [0, n_p - 1] \\
(3) \; \beta_{min} - \beta_k \le \beta^k - \beta_k \le \beta_{max} - \beta_k \; , \; \forall k \in [0, n_p] \\
(4) \; \mathbf{u}_{prev} + \Delta t \; \dot{\mathbf{u}}_{min} - \mathbf{u}_0 \le \mathbf{u}^0 - \mathbf{u}_0 \le \mathbf{u}_{prev} + \Delta t \; \dot{\mathbf{u}}_{max} - \mathbf{u}_0 \\
(5) \; \mathbf{u}_{min} - \mathbf{u}_k \le \mathbf{u}^k - \mathbf{u}_k \le \mathbf{u}_{max} - \mathbf{u}_k \; , \; \forall k \in [0, n_c] \\
(6) \; \Delta t \; \dot{\mathbf{u}}_{min} + \mathbf{u}_k - \mathbf{u}_{k+1} \le (\mathbf{u}^{k+1} - \mathbf{u}_{k+1}) - (\mathbf{u}^k - \mathbf{u}_k) \\
\qquad\qquad \le \Delta t \; \dot{\mathbf{u}}_{max} + \mathbf{u}_k - \mathbf{u}_{k+1} \; , \; \forall k \in [0, n_c - 1]
\end{cases}
\end{aligned} \tag{3.4}$$

where:

- i is the i-th discretized prediction horizon,
- k is the k-th discretization instant of the i-th discretized prediction horizon,
- $\mathbf{x}^k$ and $\mathbf{u}^k$ are the optimized variables at k-th instant, and
- $\mathbf{A}_k$ and $\mathbf{B}_k$ are the equality constraint matrices that describe the state evolution with respect to the state itself and the control, respectively.

3.4 Obstacle avoidance

The obstacle avoidance has been implemented as an additional constraint that takes into account the Euclidean distance $\sqrt{(x - x^*)^2 + (y - y^*)^2}$ of the closest points between the vehicle and the obstacle:

$$d_{min} - \sqrt{(x_k - x_k^*)^2 + (y_k - y_k^*)^2} - w_k \le \begin{bmatrix} \frac{x_k - x_k^*}{\sqrt{(x_k - x_k^*)^2 + (y_k - y_k^*)^2}} \\ \frac{y_k - y_k^*}{\sqrt{(x_k - x_k^*)^2 + (y_k - y_k^*)^2}} \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \\ \beta \\ v \\ \omega \\ w \end{bmatrix} \le +\infty \; , \; \forall k \in [0, n_p] \tag{3.5}$$

where:

- d_{min} is the minimum set distance between the vehicle and the obstacle, and
- w is the slack variable introduced to avoid infeasibility problems.

Due to the introduction of the slack variable, the objective function in (3.4) changed as follows:

$$\begin{aligned} \min_{\mathbf{x}-\mathbf{x}_i,\mathbf{u}-\mathbf{u}_i,\mathbf{w}-\mathbf{w}_i} \quad & J(\mathbf{x},\mathbf{u},\mathbf{w}) \approx J(\mathbf{x}_i,\mathbf{u}_i,\mathbf{w}_i) + \nabla J(\mathbf{x}_i,\mathbf{u}_i,\mathbf{w}_i) \begin{bmatrix} \mathbf{x}-\mathbf{x}_i \\ \mathbf{u}-\mathbf{u}_i \\ \mathbf{w}-\mathbf{w}_i \end{bmatrix} \\ & + \frac{1}{2} \left[(\mathbf{x}-\mathbf{x}_i)^T \quad (\mathbf{u}-\mathbf{u}_i)^T \quad (\mathbf{w}-\mathbf{w}_i)^T \right] \mathbf{H}J(\mathbf{x}_i,\mathbf{u}_i,\mathbf{w}_i) \begin{bmatrix} \mathbf{x}-\mathbf{x}_i \\ \mathbf{u}-\mathbf{u}_i \\ \mathbf{w}-\mathbf{w}_i \end{bmatrix} \quad , \; \forall i \in [0, n_p] \end{aligned} \tag{3.6}$$

More in detail, what happens in the program is:

- a box of points is built around the vehicle and the obstacle considering their position and dimension,
- the Euclidean distance between the two boxes is computed (Fig. 3.4 and 3.3), and
- the constraint is added to the NMPC.

Due to the simplicity of this method, strong assumptions are considered:

- the dimension of the obstacles is known,
- the trajectory followed by the dynamic obstacle is known, and
- only one obstacle per time must be avoided, otherwise it would be added one constraint for each detected obstacle which would lead to a too high computational load.

The choice of using the Euclidean distance instead of a more advanced technique, e.g. safe region (navigable space), is a consequence of what was discussed in Section 1.3; in fact, using advanced techniques (Chap. 5) makes sense if using real-sensors data, e.g. LiDAR and Camera, to estimate the dynamic obstacles trajectory with an EKF. Since this was not provided (implementing it was not possible for lack of time), and then the obstacle trajectory was given, I choose to use this simpler obstacle avoidance method. Despite this fact, this application was sufficient to achieve the purpose of testing the performance of the NMPC with both a fixed and dynamic obstacle, highlighting the computational load that this feature requires.

Note 1 In the MPC configuration file the weight of the slack variable is set strongly greater than the other decision variables in order to give it the highest priority.

Note 2 The distance is computed and kept into account along the whole prediction horizon considering the estimated vehicle and obstacle states.

3.5 Program configuration

Firstly, Model and NMPC are initialized. Then, the user's Controller uses the NMPC to solve QP sub-problems, accordingly to the SQP approach, along the set prediction horizon. Once the problem is solved, the first optimal control input is sent to the vehicle (Fig. 3.2).
Using the *multiple shooting* approach (Sec. 2.3.2), the ProxQP problem (3.2) has been configured as described in Appendix A and B respectively for the linear and nonlinear case (only the second one includes the obstacle avoidance).

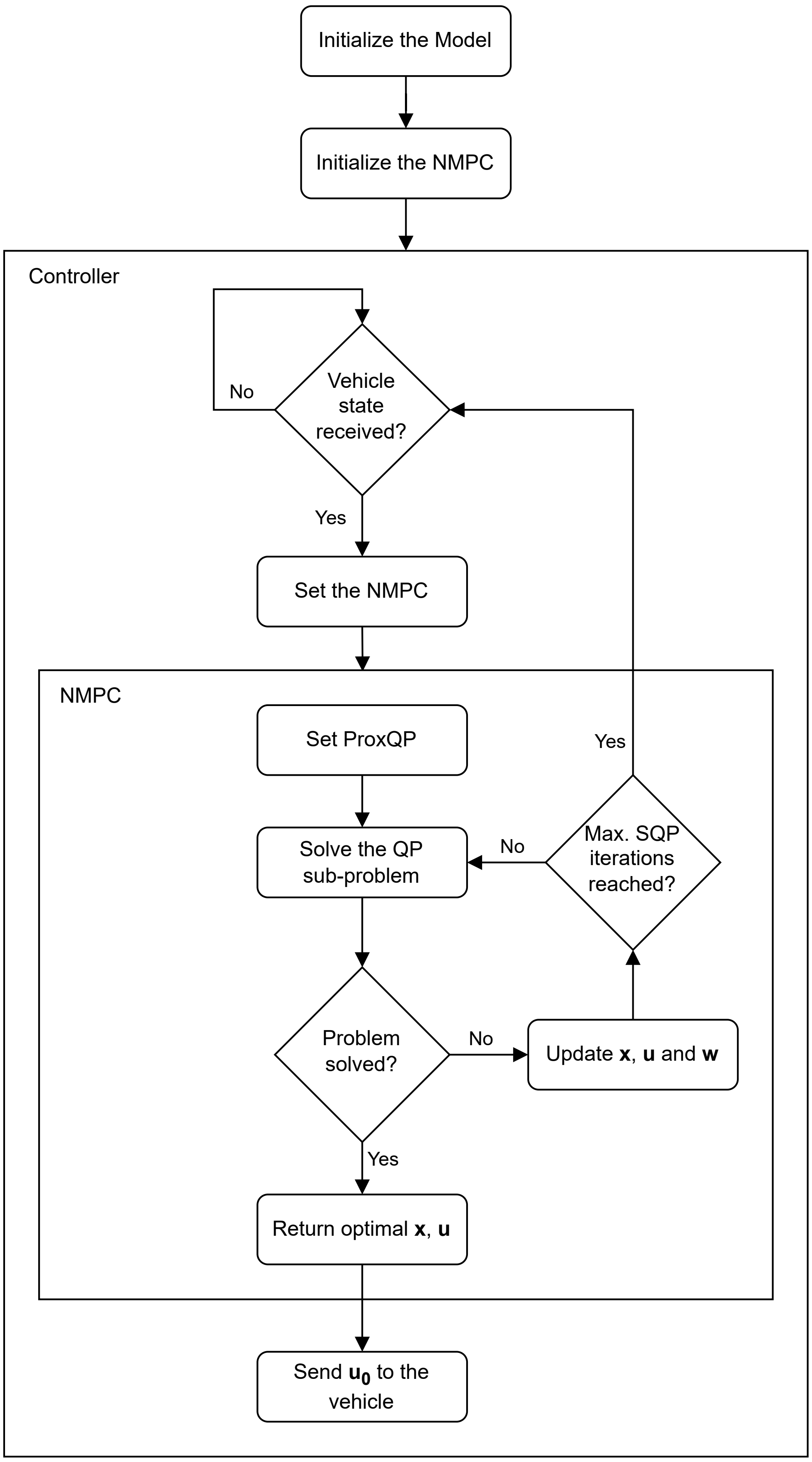


Figure 3.2: Program flowchart. The user's Controller calls the NMPC in order to send the optimal control input to the vehicle.

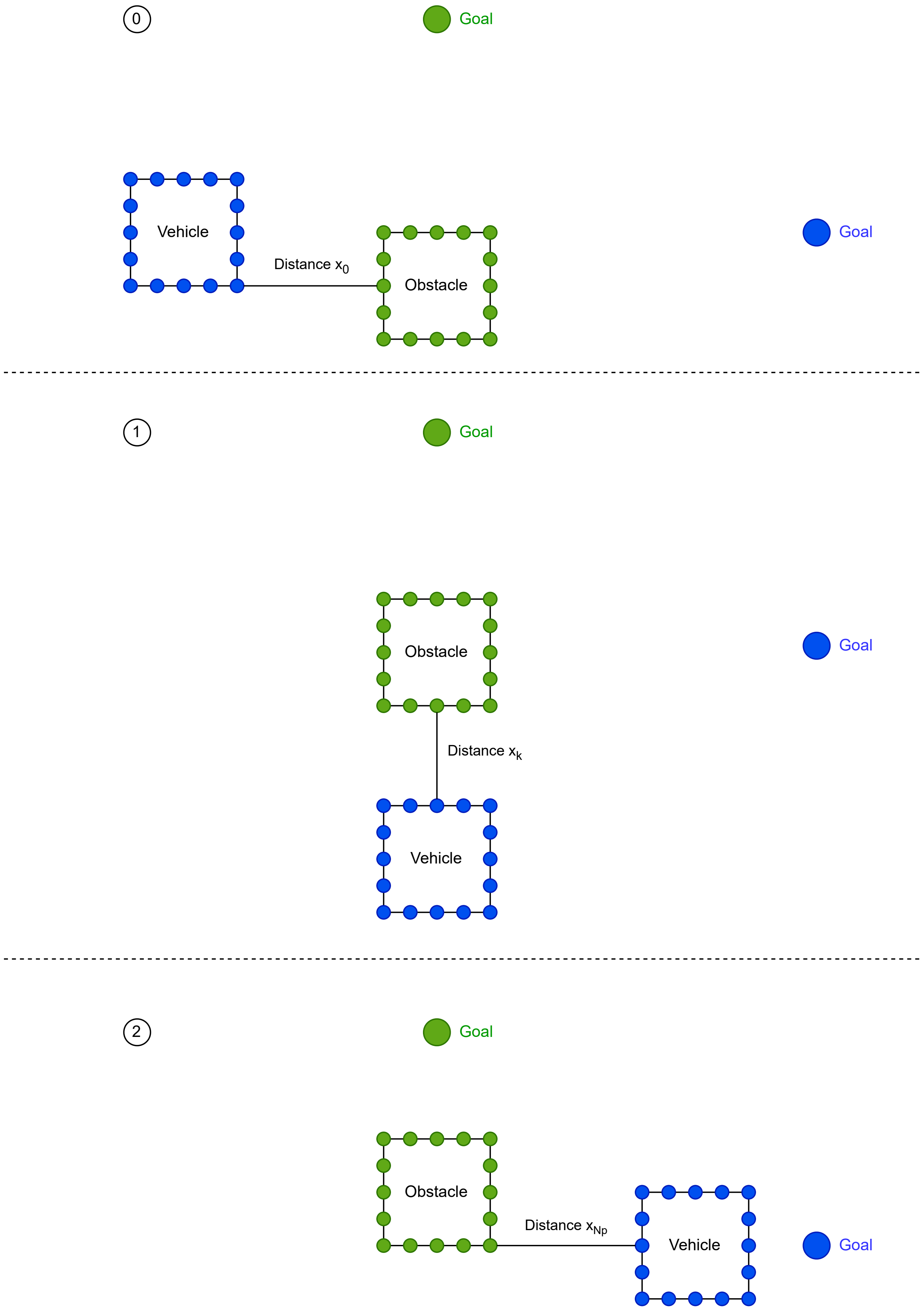


Figure 3.3: Dynamic obstacle avoidance using Euclidean distance: since the obstacle is moving towards its goal, the vehicle decides to avoid it by passing down.

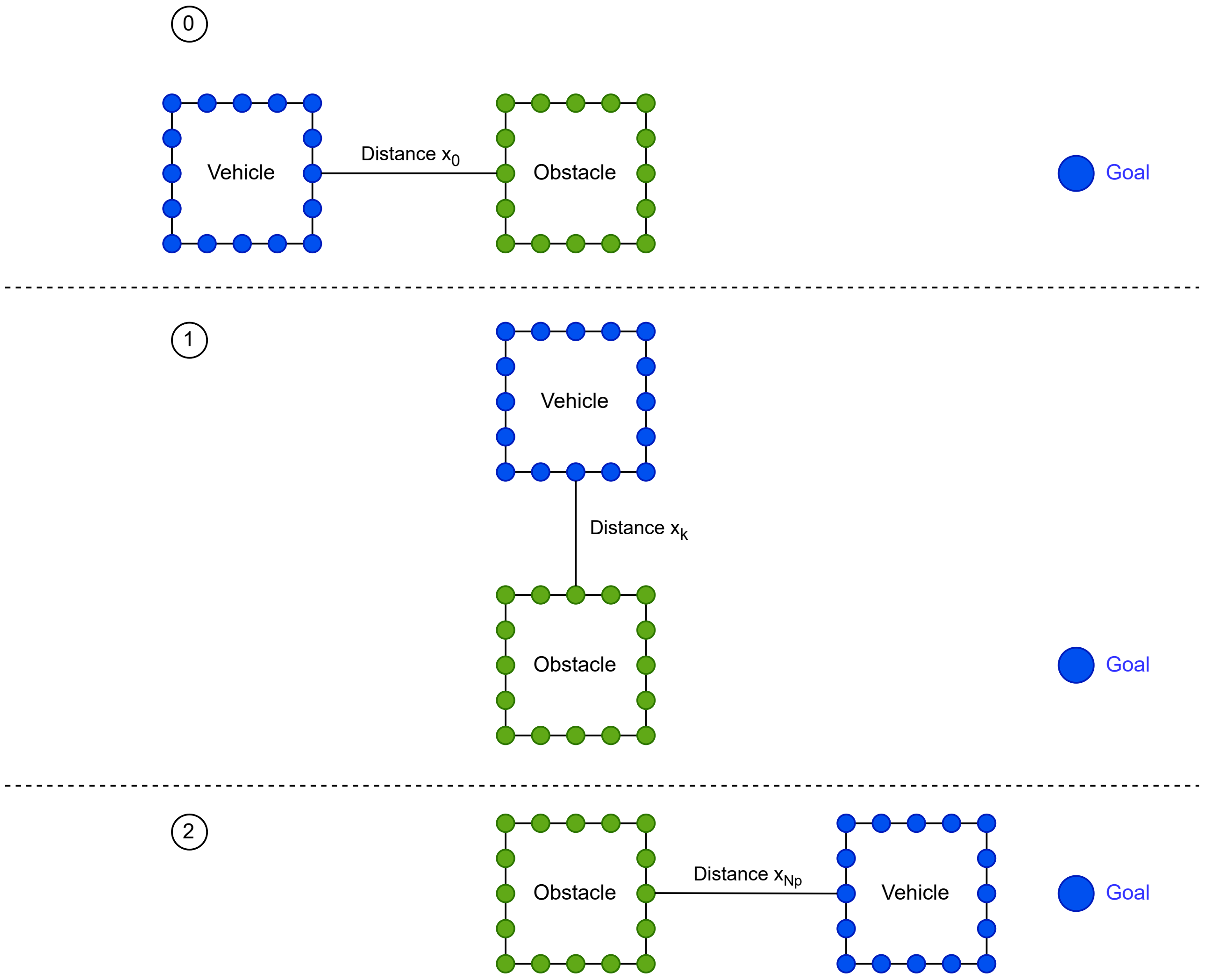


Figure 3.4: Fixed obstacle avoidance using Euclidean distance: since the obstacle is not moving the vehicle can freely avoid it passing up or down.

3.6 Measurement units

In the next sections, the measurement units for the constraints will be the following:

- x $[m]$
- y $[m]$
- θ $[rad]$
- β $[rad]$
- v $[m/s]$
- w $[rad/s]$
- $\dot{v}$ $[m/s^2]$
- $\dot{w}$ $[rad/s^2]$

3.7 Virtual environment

As a first approach, the NMPC has been tested in a virtual environment. Working in these conditions gives the possibility to test the program performances without the problems given by the real applications (e.g., message delay, slipping, skidding, etc.).
The tests have been carried out on both the unicycle (Fig. 3.1a) and the bicycle models (Fig. 3.1b).

3.7.1 Unicycle model

The unicycle model (Fig. 3.1a) was represented by the R2D2 URDF (Unified Robot Description Format) (Fig. 3.5a), with the following kinematic:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\ cos(\theta) \\ v\ sin(\theta) \\ w \end{bmatrix} \tag{3.7}$$

where:

- $[x, y, \theta]^T$ is the state vector $\mathbf{x}$ composed by the Cartesian coordinates and the orientation;
- $[v, w]^T$ is the control vector $\mathbf{u}$ composed by the linear (m/s) and the angular (rad/s) velocities.

ProxQP configuration

$$\mathbf{A}_k = \begin{bmatrix} 1 & 0 & -\Delta t\ v_k\ sin(\theta_k) \\ 0 & 1 & \Delta t\ v_k\ cos(\theta_k) \\ 0 & 0 & 1 \end{bmatrix} \quad , \ \forall k \in [0, n_p]$$

$$\mathbf{B}_k = \begin{bmatrix} \Delta t\ cos(\theta_k) & 0 \\ \Delta t\ sin(\theta_k) & 0 \\ 0 & \Delta t \end{bmatrix} \quad , \ \forall k \in [0, n_p]$$

$$\mathbf{c}_k = \begin{bmatrix} x_k - x_{k+1} + \Delta t\ v_k\ cos(\theta_k) \\ y_k - y_{k+1} + \Delta t\ v_k\ sin(\theta_k) \\ \theta_k - \theta_{k+1} + \Delta t\ w_k \end{bmatrix} \quad , \ \forall k \in [0, n_p]$$

Constraints

$$\forall k \in [0, n_c] : \begin{cases} (1) & -3.0 \leq v_k \leq +3.0 \\ (2) & -1.0 \leq \omega_k \leq +1.0 \\ (3) & -0.5 \leq \dot{v}_k \leq +0.5 \\ (4) & -0.5 \leq \dot{\omega}_k \leq +0.5 \end{cases}$$

Obstacle avoidance

- Distance: 0.2 [m]
- Vehicle box dimension: 0.5 × 0.4 [m]
- Obstacle box dimension: 0.5 × 0.4 [m]

Tuning

- $n_p = 50$
- $n_c = 10$
- $\Delta t = 0.1\ [s]$
- $\mathbf{Q} = \mathbf{S} = \mathbf{I}_{3\times3}$
- $\mathbf{R} = \mathbf{0}_{2\times2}$
- $\mathbf{W} = 1000\ \mathbf{I}_{1\times1}$

3.7.2 Bicycle model

The bicycle model (Fig. 3.1b) in the virtual environment was represented by the Bike URDF (Fig. 3.5b) [28], and described by the corresponding kinematic (3.1).

ProxQP configuration

$$\mathbf{A}_k = \begin{bmatrix} 1 & 0 & -\Delta t\ v_k\ sin(\theta_k + \beta_k) & -\Delta t\ v_k\ sin(\theta_k + \beta_k) \\ 0 & 1 & \Delta t\ v_k\ cos(\theta_k + \beta_k) & \Delta t\ v_k\ cos(\theta_k + \beta_k) \\ 0 & 0 & 1 & \Delta t\ v_k\ cos(\beta_k)/L \\ 0 & 0 & 0 & 1 \end{bmatrix} ,\ \forall k \in [0, n_p]$$

$$\mathbf{B}_k = \begin{bmatrix} \Delta t\ cos(\theta_k + \beta_k) & 0 \\ \Delta t\ sin(\theta_k + \beta_k) & 0 \\ \Delta t\ sin(\beta_k)/L & 0 \\ 0 & \Delta t \end{bmatrix} ,\ \forall k \in [0, n_p]$$

$$\mathbf{c}_k = \begin{bmatrix} x_k - x_{k+1} + \Delta t\ v_k\ cos(\theta_k + \beta_k) \\ y_k - y_{k+1} + \Delta t\ v_k\ sin(\theta_k + \beta_k) \\ \theta_k - \theta_{k+1} + \Delta t\ v_k\ sin(\beta_k)/L \\ \beta_k - \beta_{k+1} + \Delta t\ w_k \end{bmatrix} ,\ \forall k \in [0, n_p]$$

Constraints

$$\begin{cases} (1)\ -\frac{\pi}{2} \le \beta_k \le +\frac{\pi}{2}\ ,\ \forall k \in [0, n_p] \\ (2)\ -3.0 \le v_k \le +3.0\ ,\ \forall k \in [0, n_c] \\ (3)\ -1.0 \le \omega_k \le +1.0\ ,\ \forall k \in [0, n_c] \\ (4)\ -0.5 \le \dot{v}_k \le +0.5\ ,\ \forall k \in [0, n_c] \\ (5)\ -0.5 \le \dot{\omega}_k \le +0.5\ ,\ \forall k \in [0, n_c] \end{cases}$$

Obstacle avoidance

- Distance: 0.2 [m]
- Vehicle box dimension: 2.7 × 1.0 [m]
- Obstacle box dimension: 0.5 × 0.4 [m]

Tuning

- $n_p = 50$
- $n_c = 10$
- $\Delta t = 0.1$ $[s]$
- $\mathbf{Q} = \mathbf{S} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$
- $\mathbf{R} = \mathbf{0}_{2\times 2}$
- $\mathbf{W} = 1000\ \mathbf{I}_{1\times 1}$

3.8 Real environment

3.8.1 ROSbot 2R

As a second approach, the NMPC has been tested on the ROSbot 2R [29] (Fig. 3.5c), after having opportunely configured it [30], based on the unicycle model (Sec. 3.7.1).
Since the ROSbot was configured in ROS 1, the bridge [31] has been used to map the required topics. Using only the odometry with this kind of robot leads to localization issues due to drifting while turning. For this reason, the Adaptive Monte Carlo Localization (AMCL) [32] method has been used to solve the problem: using a static map previously built thanks to the SLAM algorithm and the laser scanner data provided by the endowed LiDAR, the localization was adjusted by using the transformation matrix between the odometry (local) frame and the map (global) one.

Constraints

$$\begin{cases} (1)\ -1.0 \leq y_k \leq +0.1\ ,\ \forall k \in [0, n_p] \\ (2)\ -0.5 \leq v_k \leq +0.5\ ,\ \forall k \in [0, n_c] \\ (3)\ -1.0 \leq \omega_k \leq +1.0\ ,\ \forall k \in [0, n_c] \\ (4)\ -0.25 \leq \dot{v}_k \leq +0.25\ ,\ \forall k \in [0, n_c] \\ (5)\ -0.5 \leq \dot{\omega}_k \leq +0.5\ ,\ \forall k \in [0, n_c] \end{cases}$$

Obstacle avoidance

- Distance: 0.2 [m]
- Vehicle box dimension: 0.228 $\times$ 0.197 [m]
- Obstacle box dimension: 0.5 $\times$ 0.3 [m]

Tuning

- $n_p = 50$
- $n_c = 50$
- $\Delta t = 0.2\ [s]$
- $\mathbf{Q} = \mathbf{S} = \mathbf{I}_{3\times 3}$
- $\mathbf{R} = 0.01\ \mathbf{I}_{2\times 2}$
- $\mathbf{W} = 1000\ \mathbf{I}_{1\times 1}$

3.8.2 Zoe car

As a final approach, the NMPC has been tested on the Zoe car (Fig. 3.5d) based on the bicycle model (Sec. 3.7.2). The vehicle is endowed with IMU, GPS, Camera, and LiDAR; these sensors provide real-time data that are fused together by a built-in library (*iCars*) which allows to build a point cloud map of the surrounding environment and estimate the car position using an EKF. Also in this case the ROS 1 bridge was necessary to communicate with the vehicle. Moreover, a Python converter node has been initially written to make the needed ROS 1 custom messages suitable for ROS 2, allowing the exchange of velocities and steering angle data. These ones have been used to localize the vehicle using the odometry and to send the requested control inputs (optimal linear velocity and steering angle). Later, the EKF data have been used for better localization and more consistent data tests.

Constraints

$$\begin{cases} (1)\ -\frac{\pi}{6} \le \beta_k \le +\frac{\pi}{6}\ ,\ \forall k \in [0, n_p] \\ (2)\ +0.0 \le v_k \le +3.0\ ,\ \forall k \in [0, n_c] \\ (3)\ -0.5 \le \omega_k \le +0.5\ ,\ \forall k \in [0, n_c] \\ (4)\ -0.5 \le \dot{v}_k \le +0.5\ ,\ \forall k \in [0, n_c] \\ (5)\ -0.5 \le \dot{\omega}_k \le +0.5\ ,\ \forall k \in [0, n_c] \end{cases}$$

Obstacle avoidance

- Distance: 0.2 [m]
- Vehicle box dimension: 1.73 × 4.084 [m]
- Obstacle box dimension: 0.5 × 0.3 [m]

Tuning

- $n_p = 50$
- $n_c = 50$
- $\Delta t = 0.2\ [s]$
- $\mathbf{Q} = \mathbf{S} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$

- $\mathbf{R} = 0.01\ \mathbf{I}_{2\times 2}$
- $\mathbf{W} = 1000\ \mathbf{I}_{1\times 1}$

3.9 Choices explanation

For the virtual vehicles, the velocity and acceleration constraints have been freely chosen since there were no real-case problems, e.g. drifting, but anyway, they have been set to make realistic sense. For the Bike, it was also set a steering angle constraint due to the mechanical structure. For the ROSbot 2R, the constraints have been chosen in order to limit the problem of drifting while turning. For the Zoe car, the steering constraint has been set considering the mechanical limits, while the velocity and the acceleration ones have been set to limit strong motions.
Notice that, in general, the constraints come out from trial and error tests, looking at the performance of the vehicle in the experiments.
The horizon in the virtual environment has been chosen of 5 [s] for the state and 1 [s] for the control since being in simulation even worse far predicted controls do not affect negatively the vehicle performances (only the first one is sent to it) (Fig. 3.2). In fact, in simulation, there are a lot fewer problems with data reception and synchronization which leads to better control performances with respect to real vehicles, and this gave the possibility of testing shorter control horizons decreasing the computational load.
The states and final state matrices are identities except for the steering angle in the bicycle model which has a zero weight since its orientation already influences the vehicle position and orientation (3.1).
The control input matrix was set to 0 in the simulation since virtual vehicles do not suffer strong control inputs, while it was set to 0.01 for the ROSbot 2R and the Zoe car gaining importance in real implementations.
The slack variable weight matrix was set to 1000 since it regards obstacle avoidance which is for sure the highest priority task.
For the real vehicles, the control horizon has been increased to 50, as the prediction one, to have better control, but this required a higher step size to reduce the computational load. Also, having a higher step size gives a longer prediction horizon that increased to 10 [s].
Due to the lack of space, for the Zoe car tests the fixed obstacle was moved by 0.1 [m] from the trajectory and the dynamic one by 0.5 [m]; this has been done to ensure the avoidance of the car on one decided side. For the same reason, the Zoe car followed the trajectory only forward, without turning to come back; this justifies the choice of having 0.0 $[\frac{m}{s}]$ as the minimum linear velocity.

(a) R2D2 URDF model.

(b) Bike URDF model.

(c) ROSbot 2R [29].

(d) Zoe car [33].

Figure 3.5: Virtual and real vehicles.

Chapter 4

Experiments and Results

4.1 Set up

The goal of the experiments was to test the performances of the NMPC while tracking a reference trajectory along a simple path (Fig. 4.1) in 2 different situations:

1. with a fixed obstacle, and
2. with a dynamic obstacle that follows an orthogonal straight-line path.

The vehicle runs the path 2 times, forward and backward. The obstacle has been positioned in the middle of the path without a *friendly* [15] behaviour, i.e., seeking to avoid collisions, in the dynamic case. It is important to remind (Sec. 1.3 and 3.4) that the dynamic obstacle trajectory was given and not estimated as it would be in real-case scenarios.

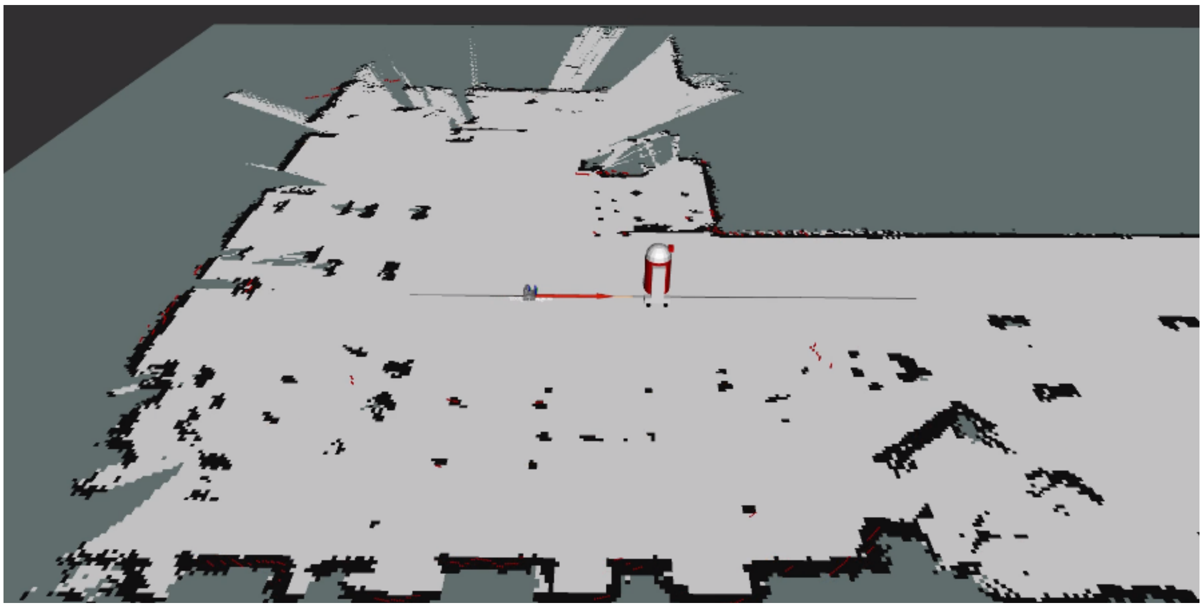


Figure 4.1: Experiments path in RViz: straight line with a fixed or dynamic obstacle.

4.1.1 Software and Hardware details

The program is coded in C++ and has been developed in ROS 2 (Robot Operating System). The experiments have been conducted on Ubuntu 20.04 LTS, using the ROS 2 Foxy version. The details about the used hardware are reported in the following list:

- CPU: Intel(R) Core(TM) i7-10750H CPU @ 2.60 GHz
- RAM: 16 Gb

4.2 Goals

The aim of the experiments was to verify and show the desired behaviour of the NMPC established by:

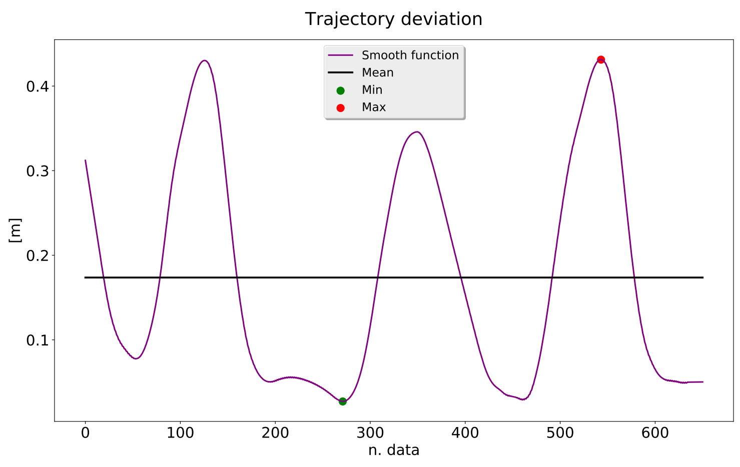
- its workability,
- the constraints satisfaction,
- the possibility of tuning (weights, horizons and step size),
- the possible implementation of an obstacle avoidance constraint, and
- the versatility due to the possibility of application to various models.

For these reasons, the obtained results are analyzed from a qualitative point of view, since a quantitative one would require a deeper development of the used controller and trajectory generation which were not the focus of the carried-out work.
To show the achievement of the listed features, in the following are reported and commented the obtained results raised by the configurations provided in Chapter 3 (Sec. 3.7 and 3.8) and the set-up described above (Sec. 4.1).
The images are divided for each model and for the fixed and dynamic obstacle cases.
The trajectory deviation, the predicted one and the solving time are grouped in a vertical disposition to better highlight the existing assonance. The constraints are grouped in a separate image.
The functions are filtered with the *Savitzky-Golay* filter, leading to a smooth shape easier readable.

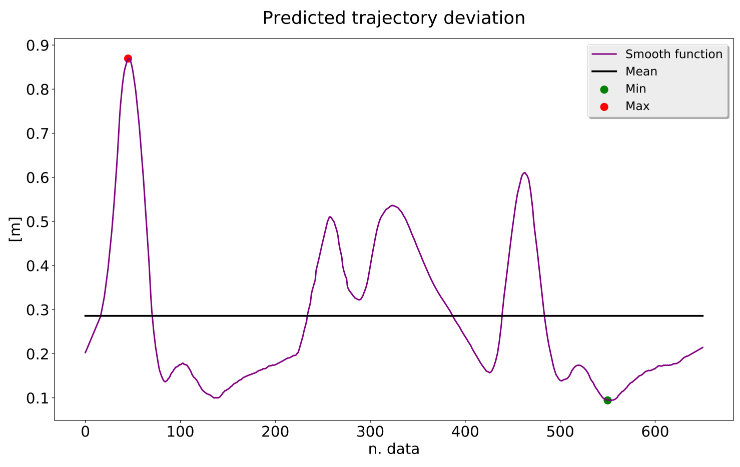
4.3 R2D2

4.3.1 Fixed obstacle

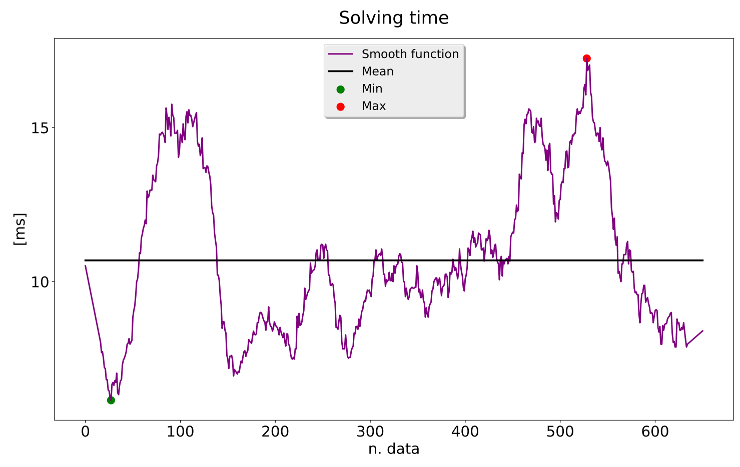
Video available.[1]



(a) Trajectory deviation.



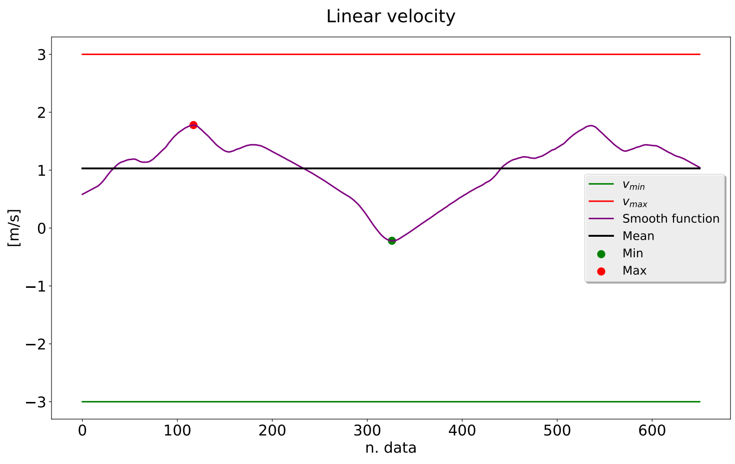
(b) Predicted trajectory deviation.



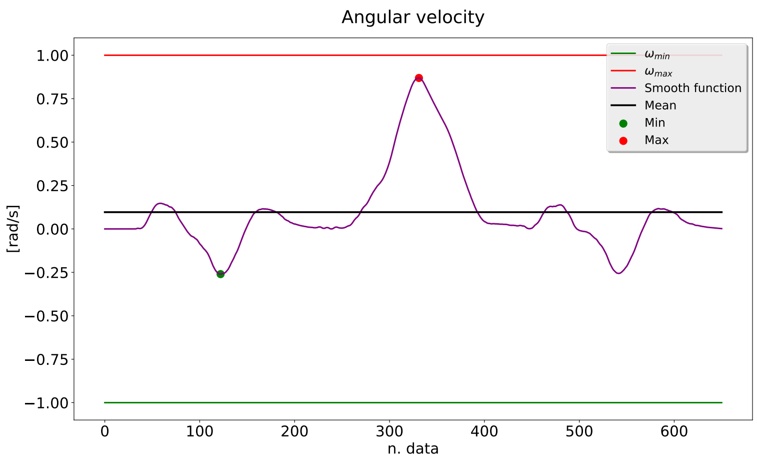
(c) Solving time.

Figure 4.2: R2D2 results with a fixed obstacle.

[1]https://youtu.be/s1l8Mkq_RpE



(a) Linear velocity.

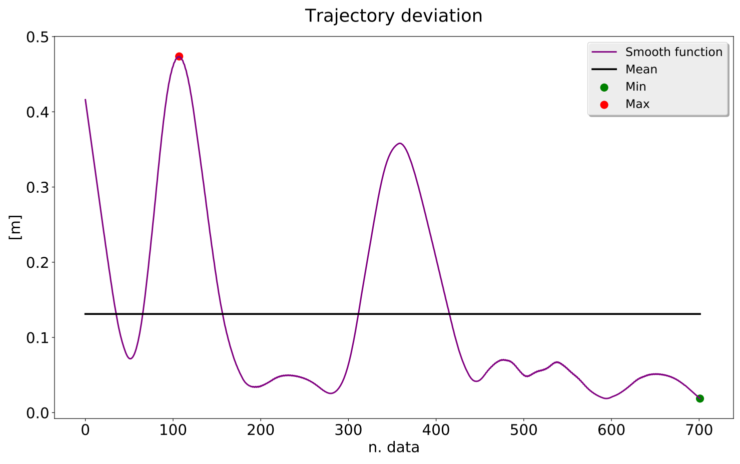


(b) Angular velocity.

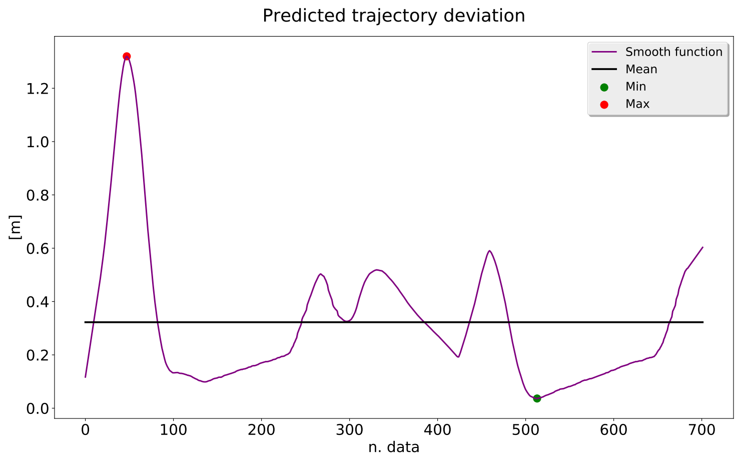
Figure 4.3: R2D2 constraints with a fixed obstacle.

4.3.2 Dynamic obstacle

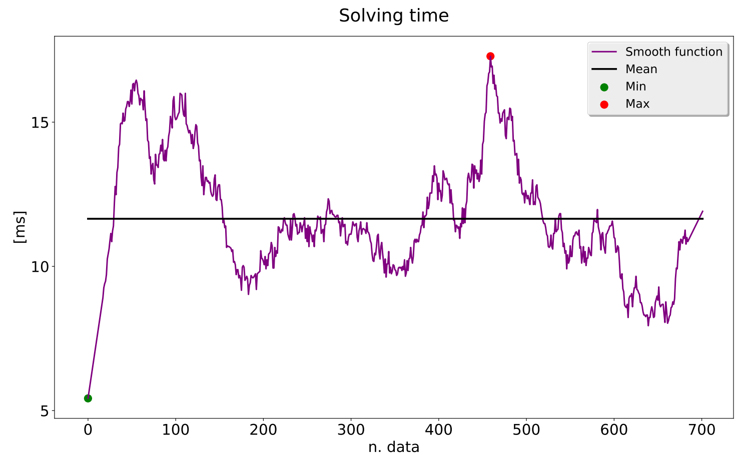
Video available.[2]



(a) Trajectory deviation.



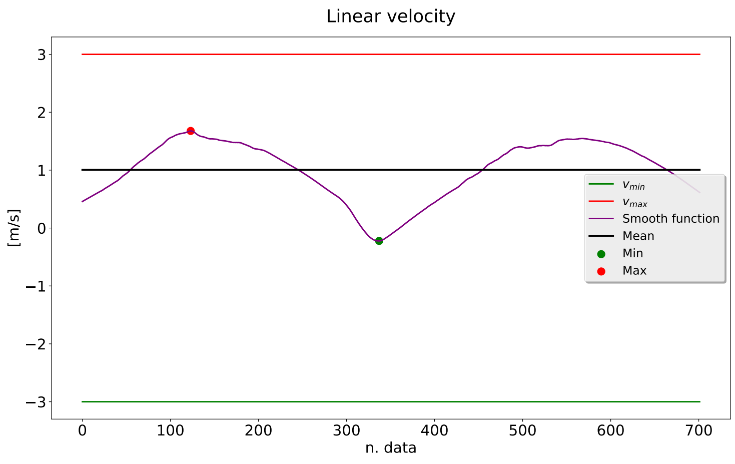
(b) Predicted trajectory deviation.



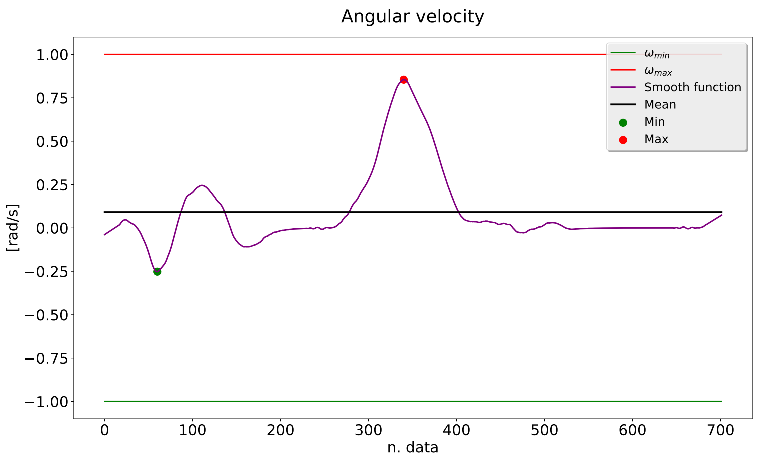
(c) Solving time.

Figure 4.4: R2D2 results with a dynamic obstacle.

[2]https://youtu.be/iZdY5_mpO60



(a) Linear velocity.



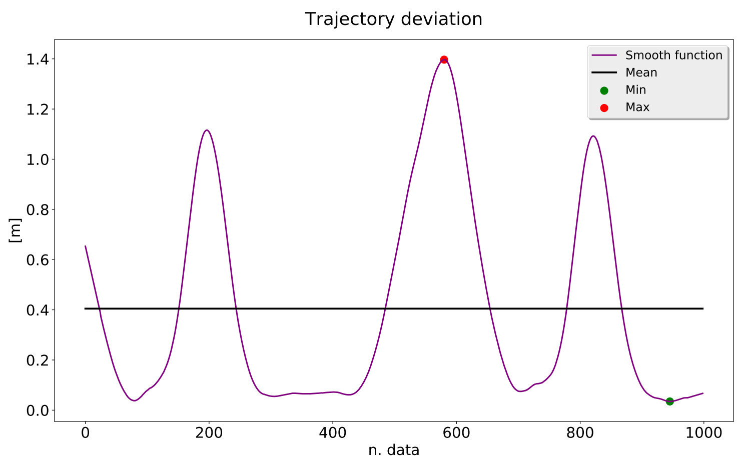
(b) Angular velocity.

Figure 4.5: R2D2 constraints with a dynamic obstacle.

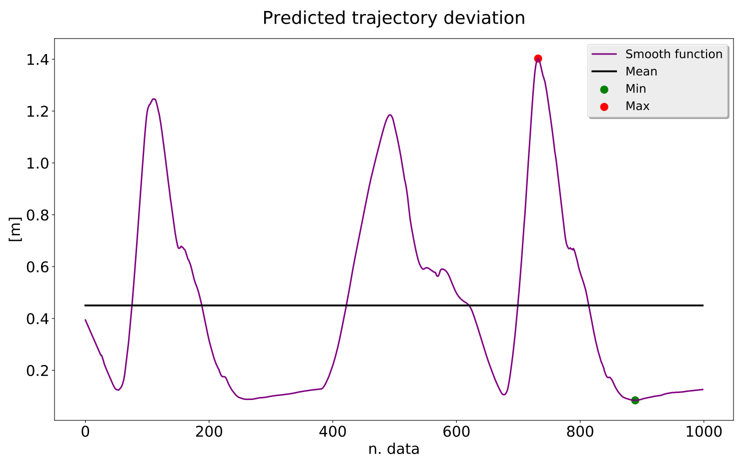
4.4 Bike

4.4.1 Fixed obstacle

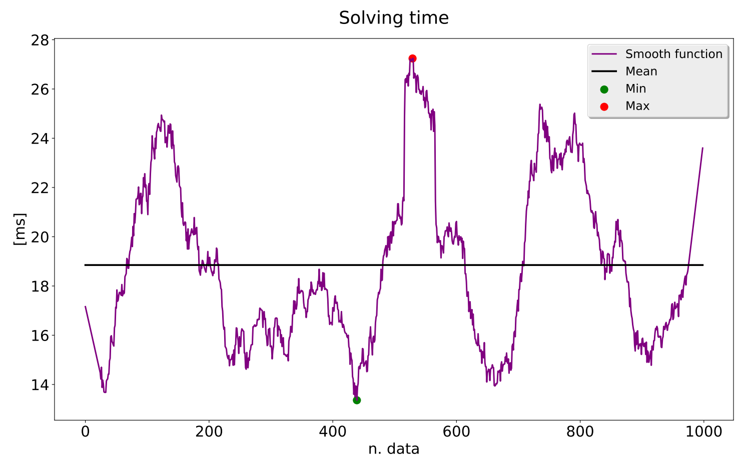
Video available.[3]



(a) Trajectory deviation.



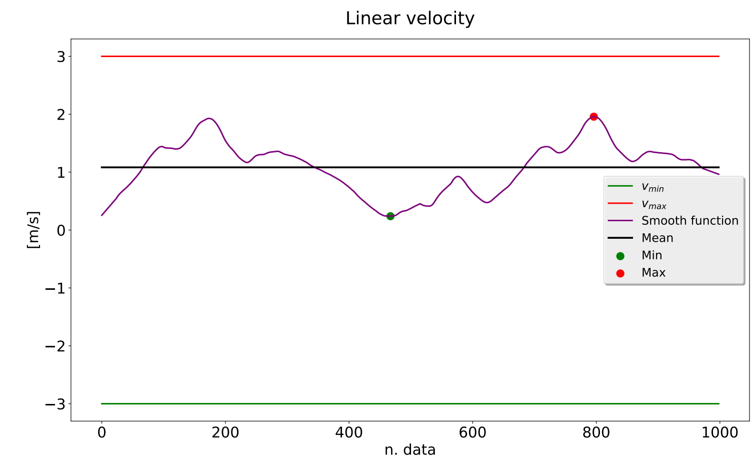
(b) Predicted trajectory deviation.



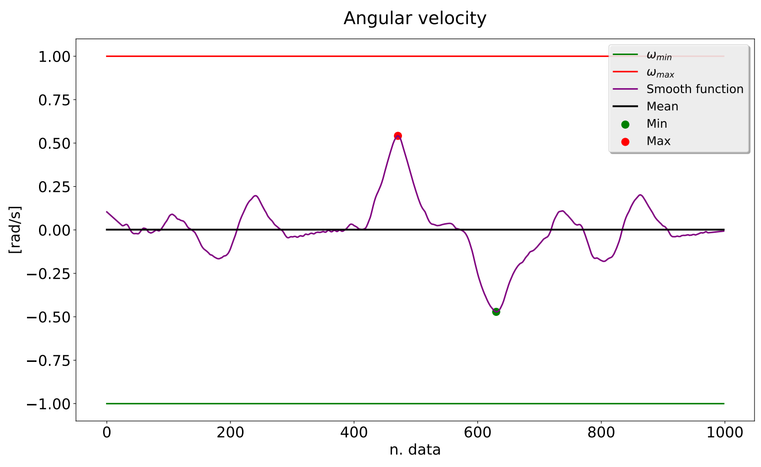
(c) Solving time.

Figure 4.6: Bike results with a fixed obstacle.

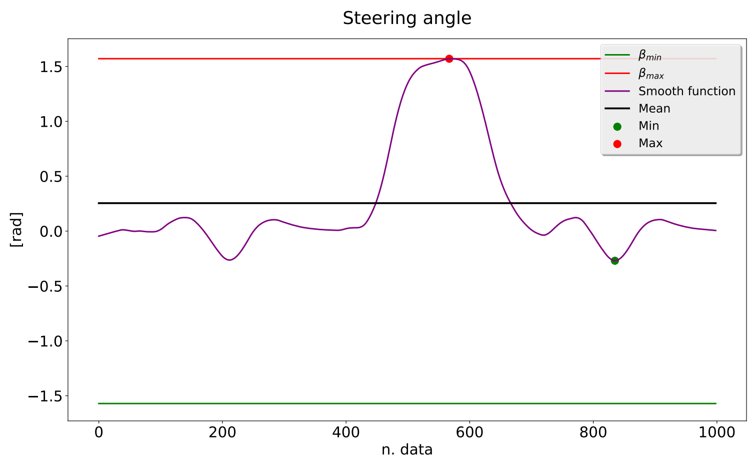
[3]https://youtu.be/9v6XMNC-qHs



(a) Linear velocity.



(b) Angular velocity.

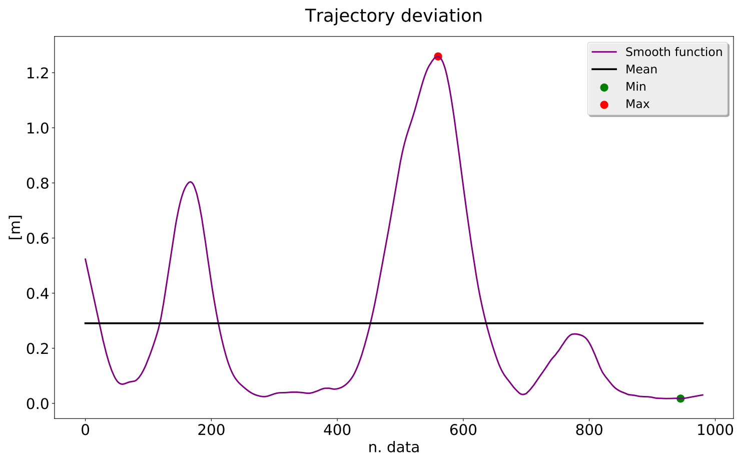


(c) Steering angle.

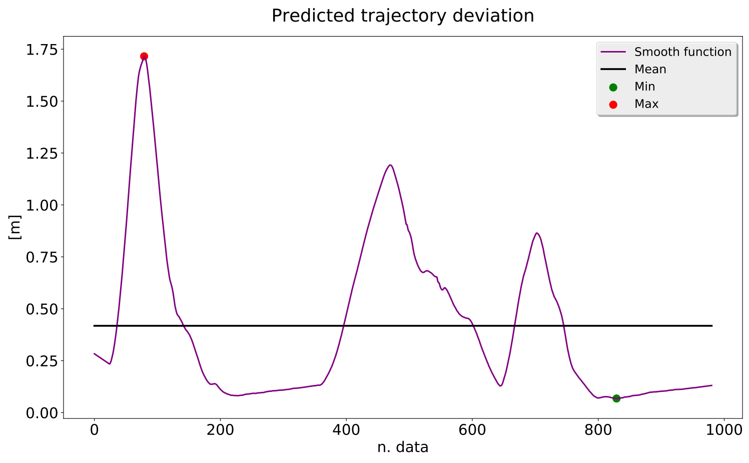
Figure 4.7: Bike constraints with a fixed obstacle.

4.4.2 Dynamic obstacle

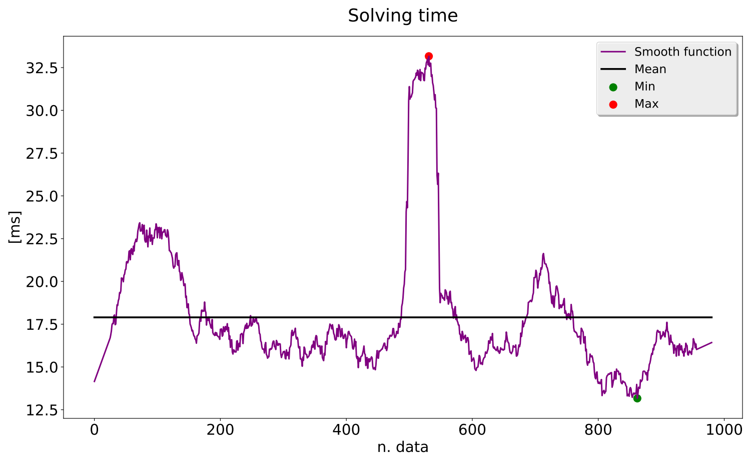
Video available.[4]



(a) Trajectory deviation.



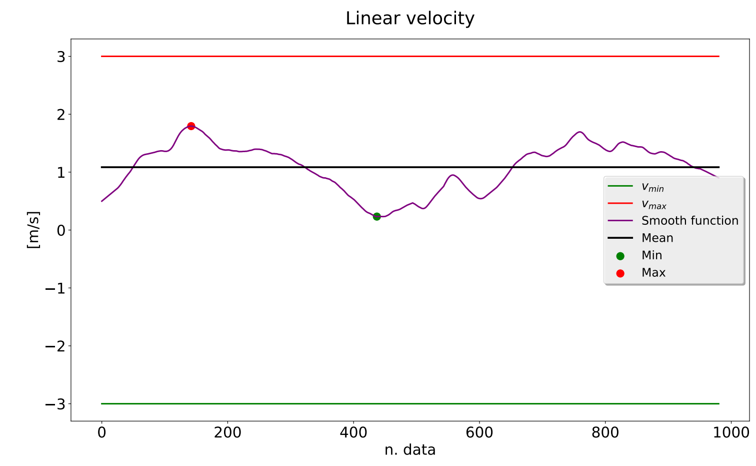
(b) Predicted trajectory deviation.



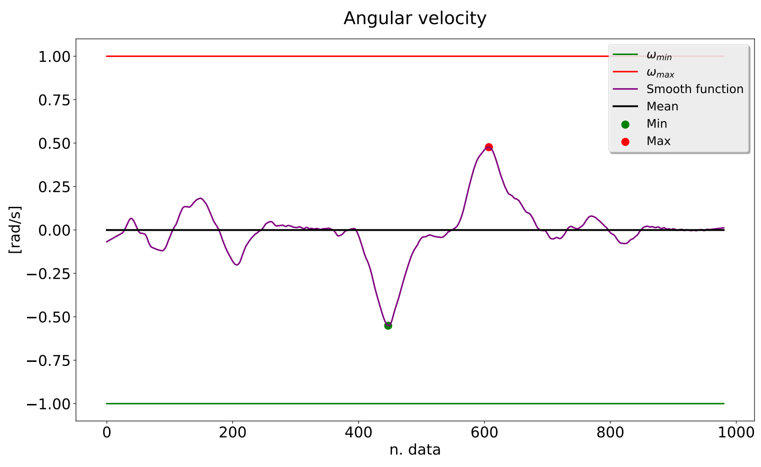
(c) Solving time.

Figure 4.8: Bike results with a dynamic obstacle.

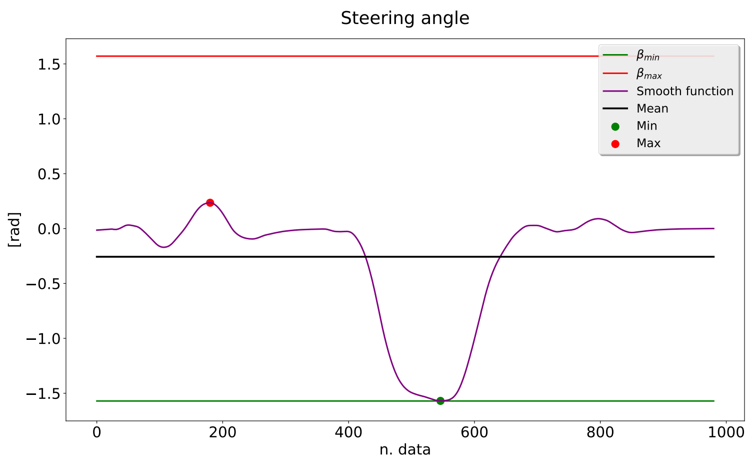
[4]`https://youtu.be/lbytIcMirI8`



(a) Linear velocity.



(b) Angular velocity.



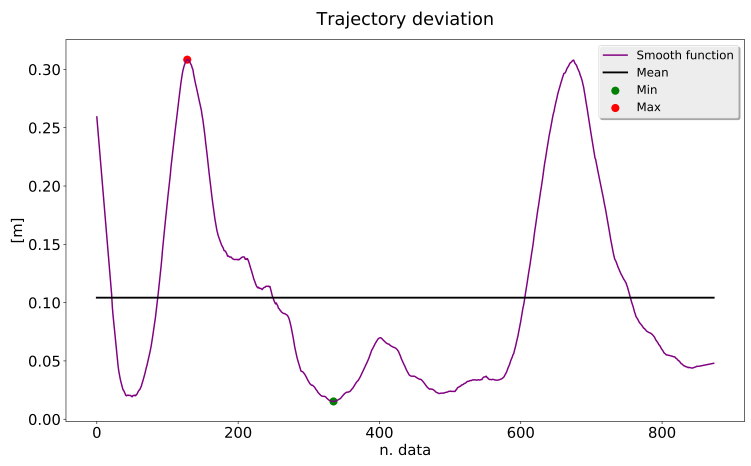
(c) Steering angle.

Figure 4.9: Bike constraints with a dynamic obstacle.

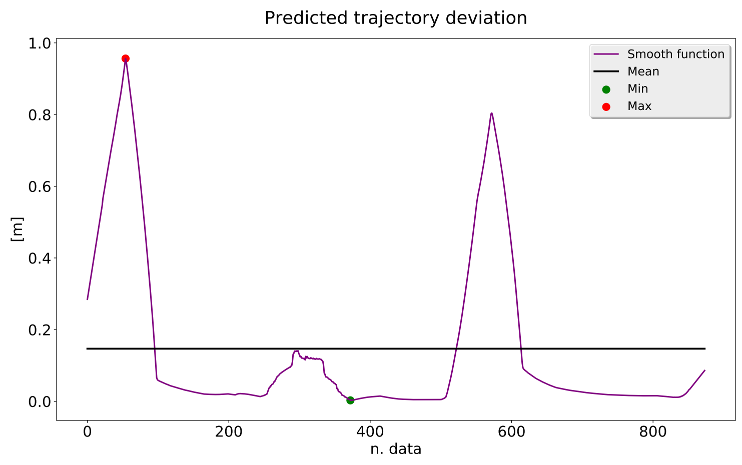
4.5 ROSbot 2R

4.5.1 Fixed obstacle

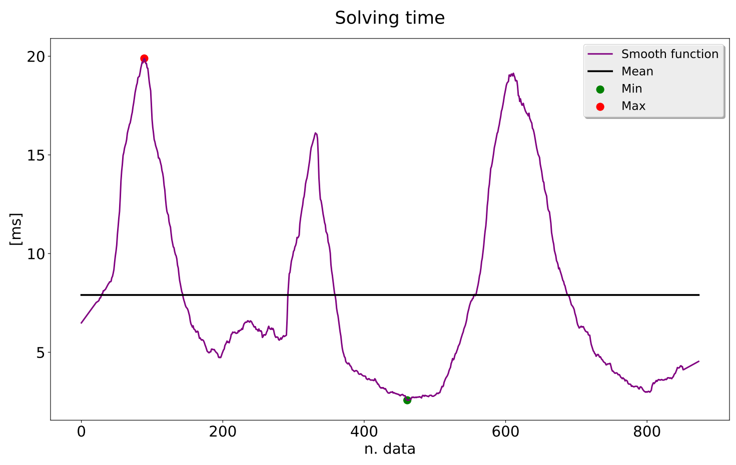
Video available.[5]



(a) Trajectory deviation.



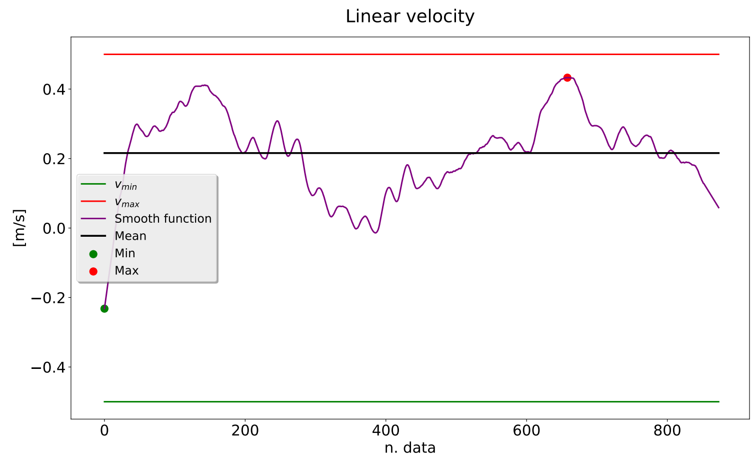
(b) Predicted trajectory deviation.



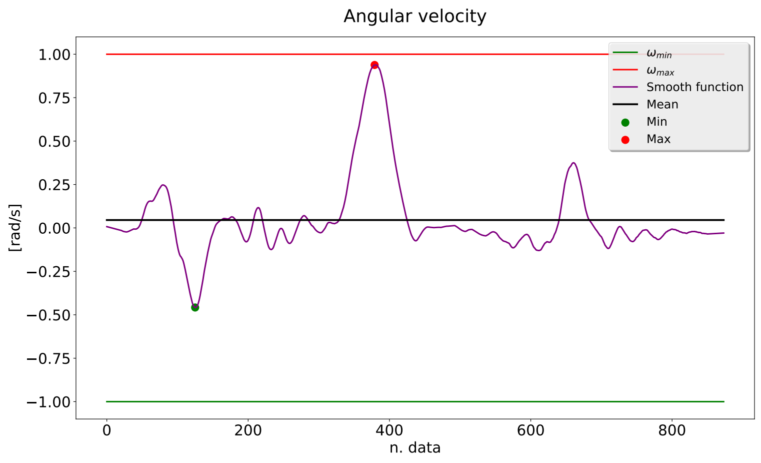
(c) Solving time.

Figure 4.10: ROSbot 2R results with a fixed obstacle.

[5]`https://youtu.be/dnS7hdsVwUI`



(a) Linear velocity.

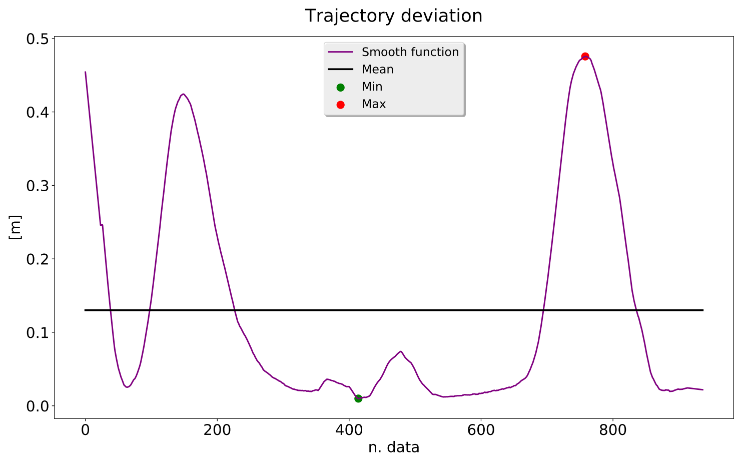


(b) Angular velocity.

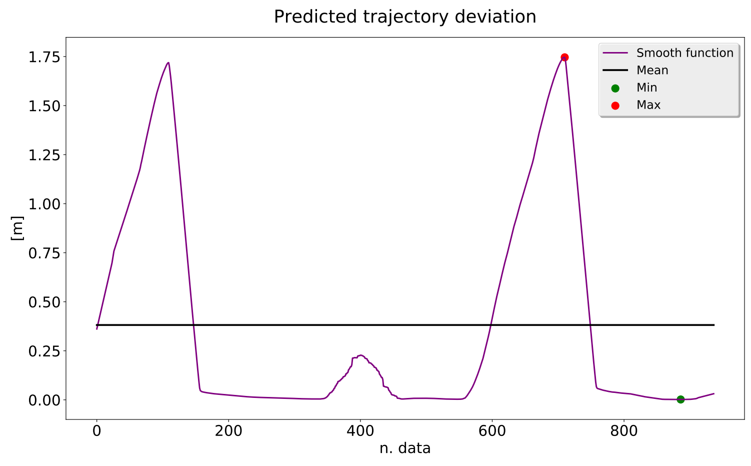
Figure 4.11: ROSbot 2R constraints with a fixed obstacle.

4.5.2 Dynamic obstacle

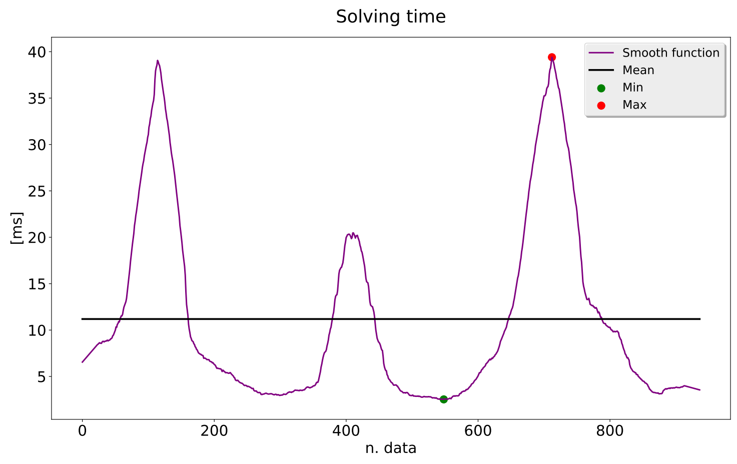
Video available.[6]



(a) Trajectory deviation.



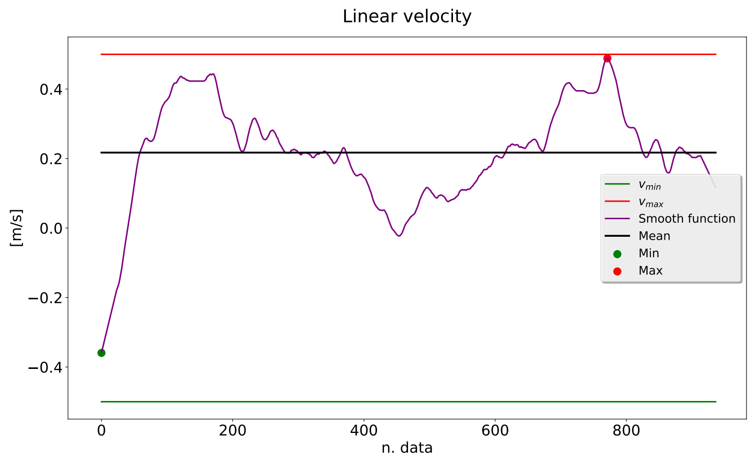
(b) Predicted trajectory deviation.



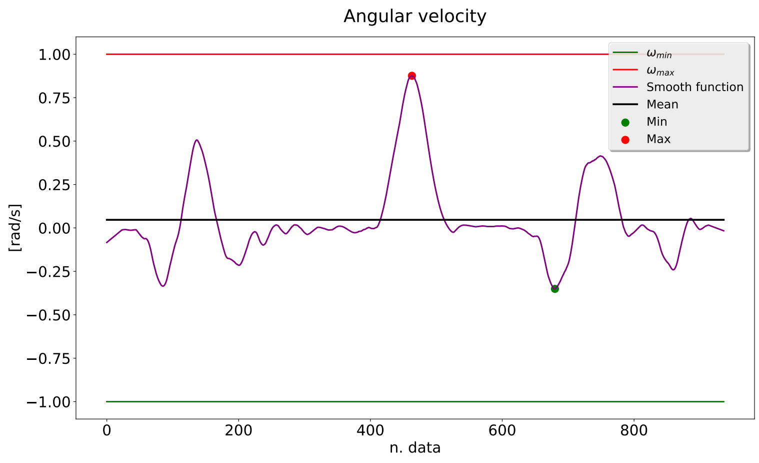
(c) Solving time.

Figure 4.12: ROSbot 2R results with a dynamic obstacle.

[6]`https://youtu.be/GNKDmMagObU`



(a) Linear velocity.

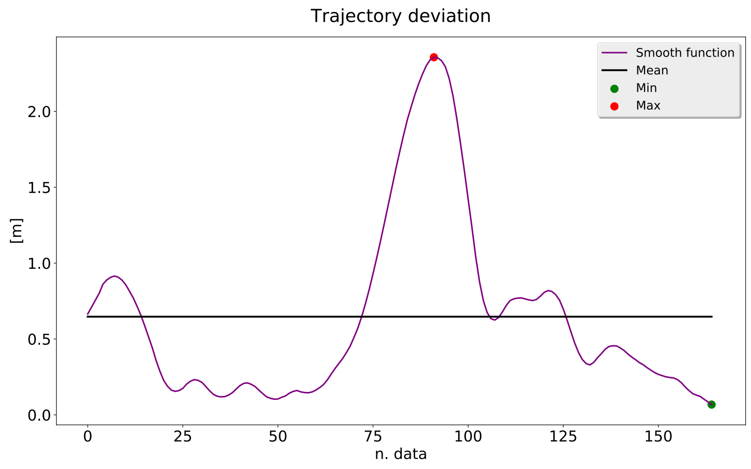


(b) Angular velocity.

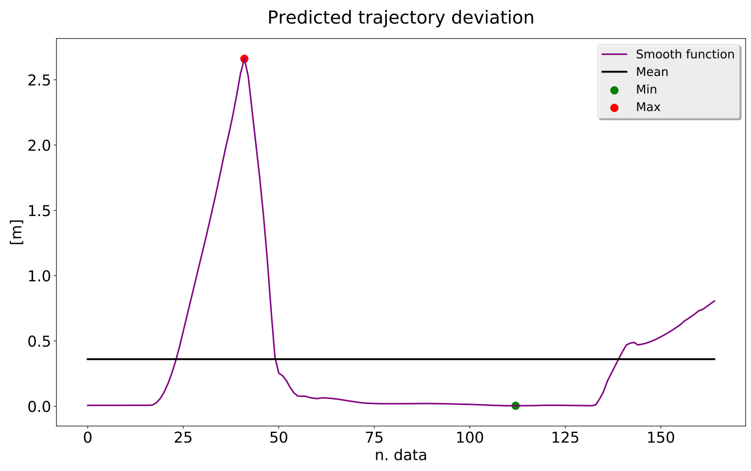
Figure 4.13: ROSbot 2R constraints with a dynamic obstacle.

4.6 Zoe car

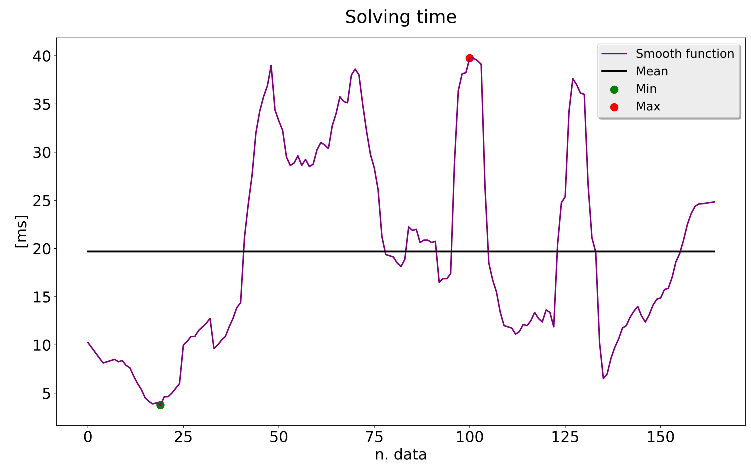
4.6.1 Fixed obstacle



(a) Trajectory deviation.

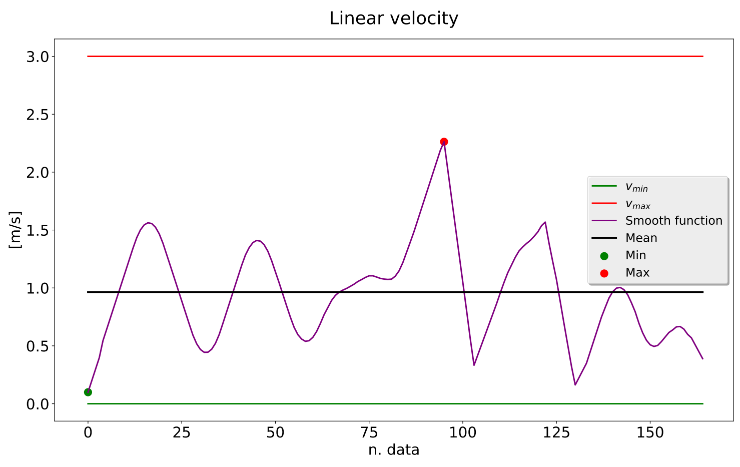


(b) Predicted trajectory deviation.

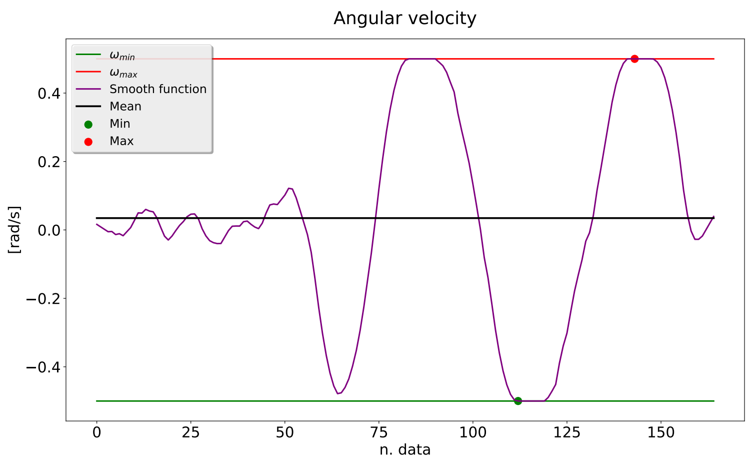


(c) Solving time.

Figure 4.14: Zoe car results with a fixed obstacle.



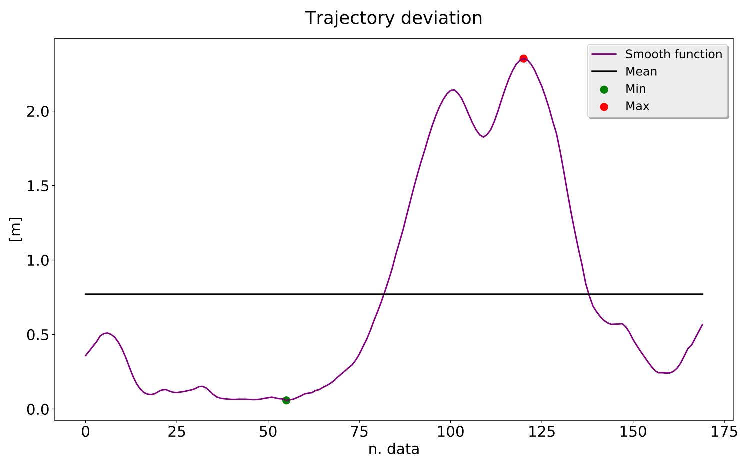
(a) Linear velocity.



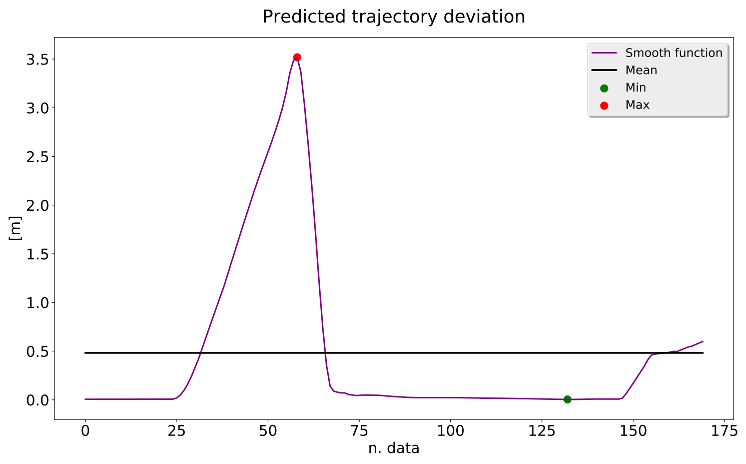
(b) Angular velocity.

Figure 4.15: Zoe car constraints with a fixed obstacle.

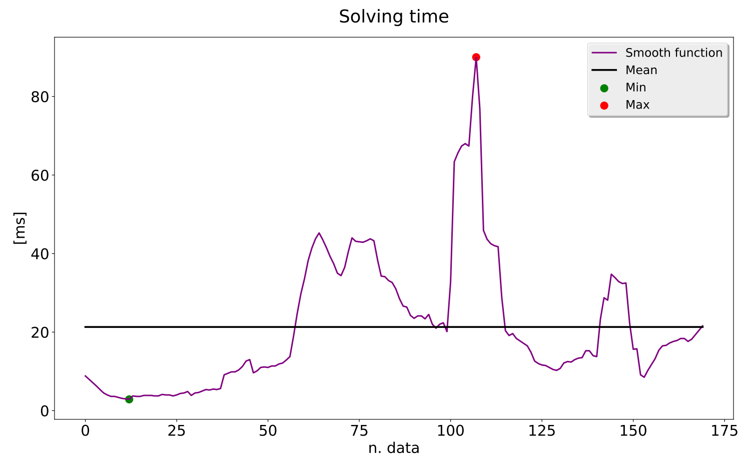
4.6.2 Dynamic obstacle



(a) Trajectory deviation.

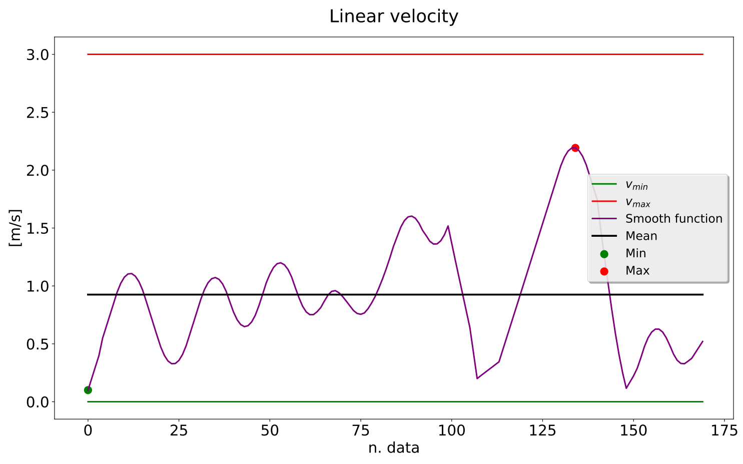


(b) Predicted trajectory deviation.

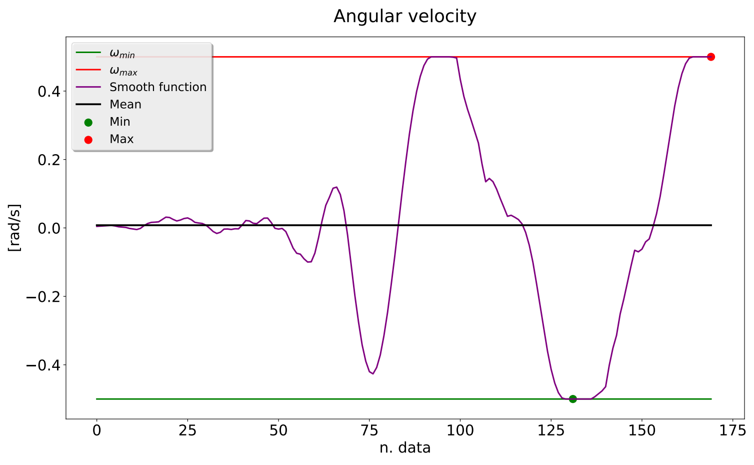


(c) Solving time.

Figure 4.16: Zoe car results with a dynamic obstacle.



(a) Linear velocity.



(b) Angular velocity.

Figure 4.17: Zoe car constraints with a dynamic obstacle.

4.7 Comments

Comparing the different cases we can see that even with different models the behaviour of the NMPC is the same:

- the trajectory deviation (Figures 4.2a, 4.4a, 4.6a, 4.8a, 4.10a, 4.12a, 4.14a, 4.16a) increases when it is necessary turning back (since we are following a line, the vehicle must turn about 180° to come back) and for sure when the obstacle must be avoided and then the vehicle must deviate from the reference trajectory;
- the predicted trajectory deviation (Figures 4.2b, 4.4b, 4.6b, 4.8b, 4.10b, 4.12b, 4.14b, 4.16b) is translated back with respect to the current one, due to the prediction feature;
- the solving time (Figures 4.2c, 4.4c, 4.6c, 4.8c, 4.10c, 4.12c, 4.14c, 4.16c) increases accordingly with the trajectory deviation (in particular the predicted one), meaning that finding a solution in case of strong turning or obstacle avoidance definitely increase the computational cost;
- the constraints (Figures 4.3, 4.5, 4.7, 4.9, 4.11, 4.13, 4.15, 4.17) are correctly satisfied, in fact in the graphs it is shown that the velocities, the accelerations, and the steering angle (for the bicycle model), remain between the set bounds.

Note In the made experiments the reference trajectory did not start on the vehicle position, this is why at the beginning the error is not 0.

4.8 Data observations

As discussed in Section 4.2, the tests had the aim of showing the NMPC performances more qualitatively than quantitatively, since the second one would require more attention on the controller and trajectory generation.
Despite this, looking to Tables 4.1 and 4.2 further observations can be done:

- the trajectory is well-tracked when it is not necessary avoiding an obstacle or doing maneuvers to turn back;
- trivially, the bigger the vehicle dimension is the greater will be the necessary deviation from the reference trajectory for the situations cited in the previous point (this justifies the bigger values for the Bike and the Zoe car);
- the more complex the model is the more time will need to solve the problem at each iteration, reaching evident peaks when moving away from the trajectory is necessary.

4.9 Considerations

The experiments highlighted the good behaviour of the NMPC, which maintains fast performances even with the presence of a fixed or dynamic obstacle while satisfying the set constraints. Also, the possibility of using the solver with different models is proof of the versatility sought. Finally, the real-world applications showed that the program is suitable also for real vehicles and not only virtual ones; this gives the vision of future improvements that can be made with the perspective of having a real-time NMPC working on different vehicles (see Chapter 5).

Vehicle	Model	Trajectory [m]			Predicted trajectory [m]			Solving time [ms]		
		Max	Min	Mean	Max	Min	Mean	Max	Min	Mean
R2D2	Unicycle	0.43	0.03	0.17	0.87	0.09	0.29	17.24	6.15	10.69
Bike	Bicycle	1.40	0.03	0.40	1.40	0.08	0.45	27.24	13.36	18.85
ROSbot 2R	Unicycle	0.31	0.02	0.10	0.96	0.00	0.15	19.89	2.57	7.90
Zoe car	Bicycle	2.36	0.07	0.65	2.66	0.00	0.36	39.75	3.75	19.70

Table 4.1: Experiments data with the fixed obstacle.

Vehicle	Model	Trajectory [m]			Predicted trajectory [m]			Solving time [ms]		
		Max	Min	Mean	Max	Min	Mean	Max	Min	Mean
R2D2	Unicycle	0.47	0.02	0.13	1.32	0.04	0.32	17.29	5.42	11.65
Bike	Bicycle	1.26	0.02	0.29	1.72	0.07	0.42	33.16	13.16	17.90
ROSbot 2R	Unicycle	0.48	0.01	0.13	1.75	0.00	0.38	39.40	2.53	11.19
Zoe car	Bicycle	2.35	0.06	0.77	3.52	0.00	0.48	90.00	2.88	21.30

Table 4.2: Experiments data with the dynamic obstacle.

Chapter 5

Possible future works

Slack variables The slack variable is implemented only for the obstacle avoidance constraint; implementing the possibility of having it for each constraint where it is considered necessary would make this functionality more versatile.

Obstacle avoidance Introducing obstacle avoidance for real applications requires real-time data for detecting both static and dynamic obstacles; then, it is necessary a Kalman Filter that predicts the states of the second ones along the prediction horizon.
Having these requirements, obstacle avoidance can be improved in this way [34]:

- take sensor data to compute a safe region (navigable space), i.e. bounds on the x and y position, through which the vehicle can freely move, and
- take into account these bounds by adding a constraint on the x and y positions in the NMPC.

Zoe car Using the approach described in the previous paragraph, the real vehicle will keep into account all the obstacles and their future states computing an optimal trajectory through the safe region. In this way, it will be possible to use the NMPC on the Zoe car at the LS2N (Fig. 3.5d), endowed with LiDAR and Camera, in real-case scenarios.

NMPCs comparison A comparison with other NMPCs present in the State of the art, or other methods, would better highlight the pros and the cons of the developed one.

Conclusion

At the beginning of this paper, the *autonomous vehicles* topic has been presented describing the internal architecture (Fig. 1) and introducing the *obstacle avoidance* task importance, which difficulties require a fast optimization problem solver to predict a feasible and safe trajectory.
In the Introduction (1) these problems have been further discussed, highlighting the autonomous navigation difficulties on the road due to the hard predictable behaviours of the driver, the other vehicles and in particular the pedestrians (Fig. 1.1) which can take decisions, e.g. crossing, very difficult to anticipate. The *Multi-Sensors-Based Predictive Control* approach [14] is an efficient solution, but its main disadvantage of falling in local minima brings to the needing of a *Nonlinear Model Predictive Control for Self-Driving Vehicles* usable in a dynamic environment.
The World model prediction (1.2) approaches have been analyzed concluding that the *conservative* one is the safest and it is made feasible by considering only the vehicle braking trajectory (*Braking Inevitable Collision States*) instead of the whole one.
In the State of the art (2), firstly the *Quadratic Programming* method has been recalled (2.1), giving its general formulation (2.1.1), analyzing both equality (2.1.2) and inequality (2.1.3) constrained cases, and discussing the hierarchical structure that can be adopted to solve conflicts between equality and inequality constraints (2.1.4).
The *Sequential Quadratic Programming (SQP)* (2.2) has been illustrated as a method to implement a NMPC. The *ProxQP* solver [3] has been presented (2.2.2) with its faster performance with respect to other QPs (Fig. 2.1).
Also the *Model Predictive Control (MPC)* has been recalled (2.3), followed by the Single (2.3.1) and Multiple (2.3.2) shooting approaches. Successively, the linear case, i.e., the *Linear Model Predictive Control (LMPC)* (2.3.3), has been briefly discussed before introducing the nonlinear case, i.e., *Nonlinear Model Predictive Control (NMPC)* (2.3.4), talking about its problems of computational load and stability. General rules [25] and possible re-formulations (e.g. *slack variables*) are given to ensure problem feasibility.
The Development and Implementation (3) have been chronologically presented, starting from the LMPC (3.2), followed by the NMPC (3.3), the Obstacle avoidance implementation (3.4), and the Program configuration (3.5). Then, the configurations for the different vehicles (virtuals and reals) used in the experiments have been provided (3.7, 3.8) with a final choices explanation (3.9).
The experiments set-up (4.1) has been shown up, and the results (4.3, 4.4, 4.5) have been provided with comments (4.7), further data observations (4.8), and final considerations (4.9) raised by their comparison. For each experiment also the respective video has been linked (except for the Zoe car due to copyright reasons).
To conclude, possible future works suggestions (5) have been given.

Appendix A

ProxQP for LMPC

A.1 Decision variables vector

$$\mathbf{z} = \begin{bmatrix} \mathbf{x}_0 \\ ... \\ \mathbf{x}_{n_p-1} \\ \mathbf{x}_{n_p} \\ \mathbf{u}_0 \\ ... \\ \mathbf{u}_{n_c} \end{bmatrix}$$

A.2 Objective function

$$\mathbf{H} = 2 \begin{bmatrix} \mathbf{Q}_0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & ... & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{Q}_{n_p-1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{S} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{R}_0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & ... & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{R}_{n_c} \end{bmatrix} \quad \mathbf{c} = -2 \begin{bmatrix} \mathbf{Q}_0 \\ ... \\ \mathbf{Q}_{n_p-1} \\ \mathbf{S} \\ \mathbf{R}_0 \\ ... \\ \mathbf{R}_{n_c} \end{bmatrix}$$

A.3 Equality constraints

$$\mathbf{E} = \begin{bmatrix} \mathbf{I}_{n\times n} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{I}_{n\times n} & -\mathbf{I}_{n\times n} & 0 & 0 & 0 & 0 & \mathbf{B}_0 & 0 & 0 \\ 0 & ... & ... & 0 & 0 & 0 & 0 & ... & 0 \\ 0 & 0 & ... & ... & 0 & 0 & 0 & 0 & \mathbf{B}_{n_c} \\ 0 & 0 & 0 & ... & ... & 0 & 0 & 0 & ... \\ 0 & 0 & 0 & 0 & \mathbf{I}_{n\times n} & -\mathbf{I}_{n\times n} & 0 & 0 & \mathbf{B}_{n_c} \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} \hat{\mathbf{x}} \\ 0 \\ ... \\ ... \\ ... \\ 0 \end{bmatrix}$$

A.4 Inequality constraints

$$\mathbf{C} = \begin{bmatrix} \mathbf{v}_{1\times n} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & ... & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{v}_{1\times n} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{I}_{m\times m} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{I}_{m\times m} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & ... & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & ... & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{I}_{m\times m} \\ 0 & 0 & 0 & -\mathbf{I}_{m\times m} & \mathbf{I}_{m\times m} & 0 & 0 \\ 0 & 0 & 0 & 0 & ... & ... & 0 \\ 0 & 0 & 0 & 0 & 0 & -\mathbf{I}_{m\times m} & \mathbf{I}_{m\times m} \end{bmatrix}$$

$$\mathbf{d}_{low} = \begin{bmatrix} \beta_{min} \\ ... \\ \beta_{min} \\ \mathbf{u}_{prev} + \Delta t \ \dot{\mathbf{u}}_{min} \\ \mathbf{u}_{min} \\ ... \\ \mathbf{u}_{min} \\ \Delta t \ \dot{\mathbf{u}}_{min} \\ ... \\ \Delta t \ \dot{\mathbf{u}}_{min} \end{bmatrix} \qquad \mathbf{d}_{upp} = \begin{bmatrix} \beta_{max} \\ ... \\ \beta_{max} \\ \mathbf{u}_{prev} + \Delta t \ \dot{\mathbf{u}}_{max} \\ \mathbf{u}_{max} \\ ... \\ \mathbf{u}_{max} \\ \Delta t \ \dot{\mathbf{u}}_{max} \\ ... \\ \Delta t \ \dot{\mathbf{u}}_{max} \end{bmatrix}$$

where $\mathbf{v}_{1\times n} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$.

Appendix B

ProxQP for NMPC

B.1 Decision variables vector

$$\mathbf{z} = \begin{bmatrix} \mathbf{x}^0 - \mathbf{x}_0 \\ ... \\ \mathbf{x}^{n_p-1} - \mathbf{x}_{n_p-1} \\ \mathbf{x}^{n_p} - \mathbf{x}_{n_p} \\ \mathbf{u}^0 - \mathbf{u}_0 \\ ... \\ \mathbf{u}^{n_c} - \mathbf{u}_{n_c} \\ \mathbf{w}^0 - \mathbf{w}_0 \\ ... \\ \mathbf{w}^{n_p} - \mathbf{w}_{n_p} \end{bmatrix}$$

B.2 Objective function

$$\mathbf{H} = 2\begin{bmatrix} \mathbf{Q}_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & ... & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{Q}_{n_p-1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{S} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{R}_0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & ... & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{R}_{n_c} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{W}_0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & ... & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{W}_{n_p} \end{bmatrix} \quad \mathbf{c} = 2\begin{bmatrix} \mathbf{Q}_0(\mathbf{x}_0 - \mathbf{x}_0^*) \\ ... \\ \mathbf{Q}_{n_p-1}(\mathbf{x}_{n_p-1} - \mathbf{x}_{n_p-1}^*) \\ \mathbf{S}(\mathbf{x}_{n_p} - \mathbf{x}_{n_p}^*) \\ \mathbf{R}_0\mathbf{u}_0 \\ ... \\ \mathbf{R}_{n_c}\mathbf{u}_{n_c} \\ \mathbf{W}_0\mathbf{w}_0 \\ ... \\ \mathbf{W}_{n_p}\mathbf{w}_{n_p} \end{bmatrix}$$

B.3 Equality constraints

$$\mathbf{E} = \begin{bmatrix} \mathbf{I}_{n\times n} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{A}_0 & -\mathbf{I}_{n\times n} & 0 & 0 & 0 & 0 & \mathbf{B}_0 & 0 & 0 \\ 0 & ... & ... & 0 & 0 & 0 & 0 & ... & 0 \\ 0 & 0 & ... & ... & 0 & 0 & 0 & 0 & \mathbf{B}_{n_c} \\ 0 & 0 & 0 & ... & ... & 0 & 0 & 0 & ... \\ 0 & 0 & 0 & 0 & \mathbf{A}_{n_p} & -\mathbf{I}_{n\times n} & 0 & 0 & \mathbf{B}_{n_c} \end{bmatrix} \quad \mathbf{b} = -\begin{bmatrix} 0 \\ \mathbf{c}_0 \\ ... \\ \mathbf{c}_k \\ ... \\ \mathbf{c}_{n_p-1} \end{bmatrix}$$

where $\mathbf{c}_k = \mathbf{x}_k - \mathbf{x}_{k+1} + \mathbf{B}_k\mathbf{u}_k$.

B.4 Inequality constraints

$$\mathbf{C} = \begin{bmatrix}
\mathbf{v}_{1\times n} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & ... & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & ... & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & ... & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & ... & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{v}_{1\times n} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \mathbf{I}_{m\times m} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \mathbf{I}_{m\times m} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & ... & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & ... & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{I}_{m\times m} \\
0 & 0 & 0 & 0 & 0 & 0 & -\mathbf{I}_{m\times m} & \mathbf{I}_{m\times m} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & ... & ... & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\mathbf{I}_{m\times m} & \mathbf{I}_{m\times m} \\
0 & \mathbf{v}_{e_0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & ... & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \mathbf{v}_{e_k} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & ... & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{v}_{e_{n_p}} & 0 & 0 & 0 & 0
\end{bmatrix}$$

$$\mathbf{d}_{low} = \begin{bmatrix}
\beta_{min} - \beta_0 \\
... \\
\beta_{min} - \beta_{n_p} \\
\mathbf{u}_{prev} + \Delta t\ \dot{\mathbf{u}}_{min} - \mathbf{u}_0 \\
\mathbf{u}_{min} - \mathbf{u}_0 \\
... \\
\mathbf{u}_{min} - \mathbf{u}_{n_c} \\
\Delta t\ \dot{\mathbf{u}}_{min} + \mathbf{u}_0 - \mathbf{u}_1 \\
... \\
\Delta t\ \dot{\mathbf{u}}_{min} + \mathbf{u}_{n_c-1} - \mathbf{u}_{n_c} \\
d_{min} - d_0 - w_0 \\
... \\
d_{min} - d_{n_p} - w_{n_p}
\end{bmatrix}
\quad
\mathbf{d}_{upp} = \begin{bmatrix}
\beta_{max} - \beta_0 \\
... \\
\beta_{max} - \beta_0 \\
\mathbf{u}_{prev} + \Delta t\ \dot{\mathbf{u}}_{max} - \mathbf{u}_0 \\
\mathbf{u}_{max} - \mathbf{u}_0 \\
... \\
\mathbf{u}_{max} - \mathbf{u}_{n_c} \\
\Delta t\ \dot{\mathbf{u}}_{max} + \mathbf{u}_0 - \mathbf{u}_1 \\
... \\
\Delta t\ \dot{\mathbf{u}}_{max} + \mathbf{u}_{n_c-1} - \mathbf{u}_{n_c} \\
+\infty \\
... \\
+\infty
\end{bmatrix}$$

where:

- $\mathbf{v}_{1\times n} = \begin{bmatrix}0 & 0 & 0 & 1\end{bmatrix}$;
- $\mathbf{v}_{e_k} = \begin{bmatrix}e_{x_k} & e_{y_k} & 0 & 0\end{bmatrix}$ with $e_{x_k} = \frac{x_k - x_k^*}{\sqrt{(x_k - x_k^*)^2 + (y_k - y_k^*)^2}}$ and $e_{y_k} = \frac{y_k - y_k^*}{\sqrt{(x_k - x_k^*)^2 + (y_k - y_k^*)^2}}$;
- $d_k = \sqrt{(x_k - x_k^*)^2 + (y_k - y_k^*)^2}$.

Bibliography

[1] M. S. Shirazi and B. Morris, "Observing behaviors at intersections: A review of recent studies & developments," in *2015 IEEE Intelligent Vehicles Symposium (IV)*, 2015, pp. 1258–1263.

[2] T. Fraichard, "Will the Driver Seat Ever Be Empty?" INRIA, Research Report RR-8493, 03 2014. [Online]. Available: https://hal.inria.fr/hal-00965176

[3] A. Bambade, S. El-Kazdadi, A. Taylor, and J. Carpentier, "PROX-QP: Yet another Quadratic Programming Solver for Robotics and beyond," in *RSS 2022 - Robotics: Science and Systems*, 06 2022, Conference Papers. [Online]. Available: https://hal.inria.fr/hal-03683733

[4] J. Drgoňa, J. Arroyo, I. Cupeiro Figueroa, D. Blum, K. Arendt, D. Kim, E. P. Ollé, J. Oravec, M. Wetter, D. L. Vrabie, and L. Helsen, "All you need to know about model predictive control for buildings," *Annual Reviews in Control*, vol. 50, pp. 190–232, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1367578820300584

[5] O. Kermorgant, "Mobile Robots - Planning and Control." [Online]. Available: https://box.ec-nantes.fr/index.php/s/bibJomHsJQbd3d6

[6] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. M. Paixão, F. Mutz, L. de Paula Veronese, T. Oliveira-Santos, and A. F. De Souza, "Self-driving cars: A survey," *Expert Systems with Applications*, vol. 165, p. 113816, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095741742030628X

[7] H. Laghmara, M.-T. Boudali, T. Josso-Laurain, J. Ledy, R. Orjuela, J.-P. Lauffenburger, and M. Basset, "Obstacle Avoidance, Path Planning and Control for Autonomous Vehicles," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, 06 2019, pp. 529–534.

[8] G. Bresson, Z. Alsayed, L. Yu, and S. Glaser, "Simultaneous Localization and Mapping: A Survey of Current Trends in Autonomous Driving," *IEEE Transactions on Intelligent Vehicles*, vol. 2, no. 3, pp. 194–220, 2017.

[9] C. Rösmann, F. Hoffmann, and T. Bertram, "Timed-Elastic-Bands for time-optimal point-to-point nonlinear model predictive control," in *2015 European Control Conference (ECC)*, 2015, pp. 3352–3357.

[10] J.-M. Park, D.-W. Kim, Y. Yoon, and K.-S. Yi, "Obstacle avoidance of autonomous vehicles based on model predictive control," *Proceedings of The Institution of Mechanical Engineers Part D-journal of Automobile Engineering - PROC INST MECH ENG D-J AUTO*, vol. 223, pp. 1499–1516, 12 2009.

[11] A. Mifsud, M. Ciocca, and P.-B. Wieber, "Comfortable and Safe Decelerations for a Self-Driving Transit Bus," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 05 2021, pp. 1005–1011.

[12] S. Contorno, "MyNMPC," 2023. [Online]. Available: https://github.com/simone-contorno/mynmpc

[13] D. A. Ridel, E. Rehder, M. Lauer, C. Stiller, and D. F. Wolf, "A Literature Review on the Prediction of Pedestrian Behavior in Urban Scenarios," *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 3105–3112, 2018.

[14] D. Pérez-Morales, O. Kermorgant, S. Domínguez-Quijada, and P. Martinet, "Multi-Sensor-based Predictive Control for Autonomous Parking in Presence of Pedestrians," in *2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2020, pp. 406–413.

[15] K. Macek, D. A. V. Govea, T. Fraichard, and R. Y. Siegwart, "Towards Safe Vehicle Navigation in Dynamic Urban Scenarios," *Automatika: Journal for Control, Measurement, Electronics, Computing and Communications*, vol. 50, pp. 184–194, 2009.

[16] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[17] T. Moore and D. Stouch, "A Generalized Extended Kalman Filter Implementation for the Robot Operating System," in *Intelligent Autonomous Systems 13*, E. Menegatti, N. Michael, K. Berns, and H. Yamaguchi, Eds. Cham: Springer International Publishing, 2016, pp. 335–348.

[18] C. G. Prevost, A. Desbiens, and E. Gagnon, "Extended Kalman Filter for State Estimation and Trajectory Prediction of a Moving Object Detected by an Unmanned Aerial Vehicle," in *2007 American Control Conference*, 2007, pp. 1805–1810.

[19] M. Herman, J. Wagner, V. Prabhakaran, N. Möser, H. Ziesche, W. Ahmed, L. Bürkle, E. Kloppenburg, and C. Gläser, "Pedestrian Behavior Prediction for Automated Driving: Requirements, Metrics, and Relevant Features," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, pp. 14 922–14 937, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:229181293

[20] A. Rudenko, L. Palmieri, M. Herman, K. M. Kitani, D. M. Gavrila, and K. O. Arras, "Human motion trajectory prediction: a survey," *The International Journal of Robotics Research*, vol. 39, pp. 895 – 935, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:155093065

[21] D. Fassbender, B. C. Heinrich, T. Luettel, and H.-J. Wuensche, "An optimization approach to trajectory generation for autonomous vehicle following," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 3675–3680.

[22] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York, NY, USA: Springer, 2006.

[23] P. D. R. H. Hoppe, *Optimization Theory.* Fall, 2006. [Online]. Available: https://www.math.uh.edu/~rohop/fall_06/

[24] M. Diehl, H. Bock, H. Diedam, and P.-B. Wieber, *Fast Direct Multiple Shooting Algorithms for Optimal Robot Control.* Berlin, Heidelberg: Springer, 2006, pp. 65–93. [Online]. Available: https://doi.org/10.1007/978-3-540-36119-0_4

[25] T. A. Johansen, "Chapter 1 Introduction to Nonlinear Model Predictive Control and Moving Horizon Estimation," in *Selected Topics on Constrained and Nonlinear Control*, 2011.

[26] D. Mayne, J. Rawlings, C. Rao, and P. Scokaert, "Constrained Model Predictive Control: Stability and Optimality," *Automatica*, vol. 36, pp. 789–814, 06 2000.

[27] F. Fusco, G. Allibert, O. Kermorgant, and P. Martinet, "Investigating the Performances of Control Parameterizations for Nonlinear Model Predictive Control," in *17th International Conference on Control, Automation, Robotics and Vision*, Singapore, Singapore, 12 2022. [Online]. Available: https://hal.science/hal-03812458

[28] T. Givan, "Functional Bicycle." [Online]. Available: https://www.thingiverse.com/thing:309158

[29] Husarion, "ROSbot 2R," Online, n.d. [Online]. Available: https://husarion.com/manuals/rosbot/

[30] M. Maragliano, "ROSbot 2R workspace," 2023. [Online]. Available: https://github.com/mmatteo-hub/rosbot2R_ws

[31] "ROS 1 bridge," 2023. [Online]. Available: https://github.com/ros2/ros1_bridge

[32] R. M. K, S. B, P. L. Uppunda, V. Raju, and C. Gururaj, "Autonomous Vehicle Navigation with LIDAR using Path Planning," 2022.

[33] "MSc Control and Robotics Advanced Robotics (CORO IMARO)." [Online]. Available: https://www.ec-nantes.fr/study/masters/advanced-robotics

[34] Turri, Valerio and Carvalho, Ashwin and Tseng, Hongtei Eric and Johansson, Karl Henrik and Borrelli, Francesco, "Linear model predictive control for lane keeping and obstacle avoidance on low curvature roads," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, 2013, pp. 378–383.