Politecnico di Milano
Corso di Laurea Magistrale in Ingegneria dell'Automazione
Dipartimento di Elettronica Informazione e Bioingegneria

POLITECNICO
MILANO 1863

Integration of model predictive control and RRTx planning algorithm for the navigation in human-crowded environments

Relatore: Prof. Luca Bascetta
Correlatore: Prof. Maria Prandini

Tesi di Laurea di:
Riccardo De Noia Matr. 890487

Anno Accademico 2022-2023

Alla mia famiglia,
che ha sempre creduto in me
più di quanto non faccia io stesso

Abstract

The work developed in this thesis belongs to the field of mobile robotics. More specifically, the focus is to develop a full navigation architecture for an autonomous wheelchair that moves within a human crowded environment. The strategy presented is based on the cooperation of two different layer, namely the Global Planner and the Controller. The Global Planner is the layer in charge of computing a collision free trajectory, considering the obstacles in the environment. The computation of the trajectory is obtained following two consecutive steps. First, a geometric path defined as a sequence of waypoints from the vehicle start to the desired goal is determined, considering the position of the obstacles and their motion. This step is achieved exploiting the RRTx algorithm, that allows to obtain an asymptotically optimal path considering a prediction of human motion within a specific time range. Once the optimal path is obtained, the waypoints are interpolated through a given time law to obtain the required trajectory.
The Controller is then responsible of solving the trajectory tracking problem, while performing obstacle avoidance, so to adapt to changes in the environment that were outside the prediction frame in the planning phase and avoid collisions, so to ensure safety for pedestrians. Hence, the vehicle might need to deviate from the planned trajectory. Then, a re-planning action is activated whenever the tracking error becomes higher than a predefined maximum value, to generate a new optimal trajectory that leads the robot to its destination.
Simulation results show the effectiveness of the solution proposed.

Sommario

Il lavoro sviluppato in questa tesi appartiene al campo della robotica mobile. Nello specifico, l'obiettivo è sviluppare un'architettura di navigazione per una carrozzina a guida autonoma che si muove all'interno di un ambiente affollato. La strategia presentata si basa sulla cooperazione di due diversi livelli, vale a dire il Pianificatore Globale e il Controllore. Il Pianificatore Globale è il livello incaricato di calcolare una traiettoria priva di collisioni, considerando gli ostacoli presenti nell'ambiente circostante. Il calcolo della traiettoria si ottiene seguendo due passi consecutivi. Innanzitutto, viene determinato un percorso geometrico definito come una sequenza di punti di passaggio intermedi, dalla posizione di partenza del veicolo verso la meta desiderata, considerando la posizione degli ostacoli e il loro movimento. Questo passaggio è ottenuto sfruttando l'algoritmo RRTx, che permette di ottenere un cammino asintoticamente ottimo considerando una previsione del movimento dei pedoni in uno specifico intervallo di tempo. Una volta ottenuto il percorso ottimo, i punti di passaggio intermedi vengono interpolati attraverso una determinata legge temporale per ottenere la traiettoria richiesta. Il controllore si occupa di risolvere il problema di inseguimento della traiettoria pianificato, garantendo l'evitamento degli ostacoli, in modo da adattarsi ai cambiamenti nell'ambiente che erano al di fuori dell'intervallo di predizione nella fase di pianificazione ed evitare le collisioni, garantendo quindi la sicurezza dei pedoni. Pertanto, il veicolo potrebbe dover deviare dalla traiettoria pianificata. Ogni qual volta si verifica la condizione per cui l'errore di tracking diventa superiore ad un valore massimo predefinito, viene attivata un'azione di ripianificazione, per generare un nuovo percorso ottimale. I risultati delle simulazioni mostrano l'efficacia della soluzione proposta.

Contents

List of Figures

Chapter 1

Introduction

In the last decades, autonomous mobile robotics applications have been largely used in many fields, such as manufacturing, agricultural production and warehouses, due to their versatility related to the capability of these systems to solve many complex tasks removing human intervention. However, in the last few years these robots are often used in more challenging contexts, that require to move within an environment populated by humans. The work done in this thesis focuses on the navigation of autonomous vehicles, more specifically Autonomous Personal Mobility Vehicles (APMV). The usage of APMV for people with motion disabilities or neurological diseases allows to provide a better assistance and facilitate many daily tasks; however, this usage might require the vehicle to navigate within crowded environment in a way that not only ensures the avoidance of all the surrounding obstacles, but that is also socially accepted by people moving around it.
Hence, the ability of autonomous vehicles to move safely and efficiently becomes important, and this necessity underscores the significance of development efforts aimed at efficient collision avoidance strategies, that also include constraints related to social conventions and human comfort. Moreover, planning and control play a relevant role in the autonomous navigation task, as well as the perception of the surrounding environment, so that the vehicle is capable of detecting changes in the environment, and hence react accordingly.

1.1 Aim of the thesis

The aim of this thesis is to present and describe the implementation of a navigation architecture for an Autonomous Personal Mobility Vehicle (an autonomous wheelchair) moving within a crowded environment, based on the cooperation of two layers: a Global Planner and a Controller. The Global Planner is the layer specialized in generating a collision-free trajectory for the vehicle. This is achieved by first computing a geometric path composed of a sequence of

waypoints, including then time information to generate the desired trajectory. The planning task is performed exploiting the RRTX algorithm [7], that allows to generate an asymptotically optimal collision-free geometric path based on the state of the environment at a specific time instant, and considering a model of future human positions within a predefined time interval.
The controller, obtained through the implementation of the Model Predictive Control (MPC) strategy, based on a linearized model obtained through the Feedback Linearization technique, is then responsible for the trajectory tracking problem, i.e. it provides the control actions required to follow the planned trajectory. The controller solves an optimization problem, subject to constraints related to vehicle performances (velocity and acceleration), as well as position constraints that limit the vehicle navigation space, to perform avoidance maneuvers accounting for human safety and comfort. Hence, the vehicle is capable of adapting to changes in the environment outside the Global Planner range of prediction for pedestrian motion, performing then socially compliant navigation.
Moreover, the two layers are interlinked, so that the Global Planner can generate a new optimal trajectory for the vehicle, whenever it needs to deviate from the previously planned one. More specifically, the tracking error is used to determine whether a re-planning action is needed. The results obtained from simulations then show the effectiveness of the cooperation of the two layers involved.

1.2 Thesis structure

In this section, the thesis structure is presented:

- Chapter 2 presents the proposed navigation architecture, obtained combining a global planning layer with a control layer, based on the Model Predictive Control method;
- Chapter 3 describes the structure of the RRTx algorithm implemented in the global planning layer, and the motion model adopted to allow the algorithm to properly compute a path avoiding possible collisions with moving obstacles in the environment;
- Chapter 4 defines the autonomous vehicle kinematic model and the linearization process that allows to obtain a linear model suitable for the control problem formulation. Furthermore, this chapter deals with the definition of the human Personal Space and Virtual Box.
- Chapter 5 describes the obstacle avoidance algorithm , based on the pedestrian model described in Chapter 4. The method described allows to define an obstacle-free region of space for the vehicle as a convex region of space. Then, the MPC formulation for the obstacle avoidance and trajectory tracking problem is presented;

- Chapter 6 shows the results obtained from simulations of the system within a crowded environment, justifying the implementation choices related to control parameters;
- Chapter 7 includes the conclusions and considerations based on the results obtained from simulations.

Chapter 2

Autunomous navigation architecture

When moving towards the desired destination, the vehicle needs to consider the environment in which it is moving, so to avoid obstacles that might occlude its path to the goal. While avoiding fixed known obstacles can be relatively simple, adapting to obstacles moving in the environment to avoid collisions can be quite challenging. In these situations, the vehicle needs to apply the control action that allows to avoid the moving obstacle, while still trying to follow the desired trajectory. If the obstacles are humans, the vehicle should move in the environment and perform avoidance maneuvers in a human-friendly way.
In such conditions, an efficient planning is required, together with a proper control strategy that allows the vehicle to move in the environment in a way that is socially accepted, still trying to reach the goal..
The aim of this chapter is to describe the cooperation between the two modules of the navigation architecture, the Global Planner and the Local Planner (or the Controller), introducing their specific role in the overall system behaviour.

2.1 Proposed navigation strategy

In this thesis, an approach that uses two different layers is adopted, to guarantee the correct navigation of the wheelchair. The first layer, specialized in planning an optimal trajectory for the vehicle given the state of the environment (fixed and moving obstacles) at the current time instant, is the Global Planner. The solution presented in this work takes the name of RRTX, presented in [7], a sampling-based approach that allows to obtain an asymptotically optimal, collision-free path between two nodes of a graph representing the navigation environment, through the knowledge of the current position of obstacles and a prediction of their future motion.

The second layer, the Controller, based on the Model Predictive Control approach, solves an optimization problem related to trajectory tracking, to provide the optimal control action to follow the desired trajectory, considering the presence of obstacles around the vehicle, to perform avoidance maneuvers in order to avoid collisions.
To do so, a region of collision-free vehicle configurations is constructed, at any time step, through the definition of linear inequalities that split the environment into two convex subspaces of the environment, so to constrain the motion of the wheelchair to the obstacle free one.
The Global Planner generates a collision-free reference trajectory inside the navigation environment based on the knowledge of the current position of the wheelchair and the position of surrounding obstacles. Given the start position P_{start} and the destination P_{goal}, the generation of such trajectory is performed in two steps:

- finding a collision-free path, expressed as a sequence of waypoints between P_{start} and P_{goal};
- interpolating the obtained waypoints through a Trapezoidal Velocity Profile.

The use of a time law between the waypoints allows to include actuation limits, hence obtaining a feasible trajectory for the vehicle.

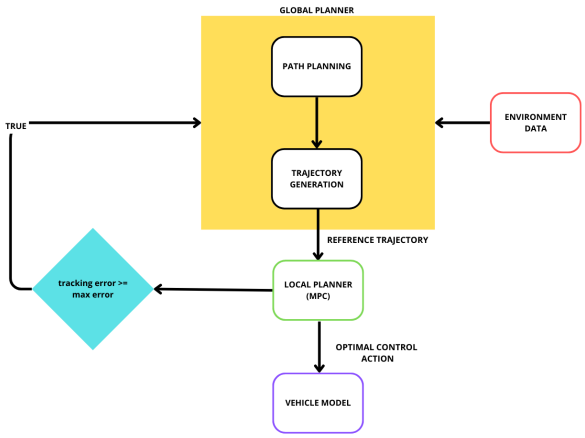


Figure 2.1: Navigation architecture

The Controller needs then to track the obtained trajectory, with a desired velocity profile, performing subtle avoidance maneuvers, that might be required during navigation in a crowded dynamic environment.
By first introducing the tracking error ε_{tr}, the overall navigation procedure, and

the cooperation between the two described modules, can be then summarized as follows:

- when the navigation starts, the Global Planner generates a first collision-free trajectory, according to the current state of the environment and considering both static and moving obstacle that are detected by the vehicle at the current time instant, depending on the range of on-board sensors;
- the Controller needs then to control the vehicle in a safe way, adapting to changes of the surrounding environment that could not be predicted by the Global Planner, while trying to track the planned trajectory;
- it might happen that, to avoid an approaching obstacle, the vehicle needs to deviate excessively from the planned trajectory, that becomes impossible to follow. In this condition, the Global Planner updates the information about the environment and the current vehicle position, and generates a new collision-free trajectory to the goal. More specifically, given the tracking error ε_{tr} computed between the current point on the ideal trajectory planned and the vehicle position, ε_{tr} is computed at any controller sample time and if it becomes higher then a predefined maximum error value ε_{max}, a new trajectory is planned, considering the new position of the obstacles in the environment, and their future predicted position.

From figure 2.1 it is clear how the knowledge of the state of the environment is crucial in both the phases of the described navigation strategy, to properly perform the planning, tracking and avoidance tasks. In the real world, such knowledge is provided by sensor readings mounted on the system that, through a data processing phase, provides data in a format that is compatible to the specific module (i.e. the Global Planner or the Controller).

Chapter 3

Global planner

3.1 RRTX path planning approach

In the field of mobile robotics, the process of path planning plays a relevant role for the navigation task of an autonomous system; indeed, thinking of a navigation problem within a crowded environment, the robot should not only reach a desired goal, but it needs to be aware of obstacles around it and avoid them. Despite the Controller will mainly take charge of the avoidance task for moving obstacles, a Global Planner method that determines the optimal path between two points in the environment, as the shortest path of a tree, is now presented, following the work done in [10] and [9]. In this chapter, a brief introduction to the path planning problem in a dynamic environment is presented, followed by the description of the RRTx algorithm implemented. Then, the motion model used to include pedestrian motion predictions in the path planning algorithm is described, so to make the vehicle able to navigate in a human crowded environment avoiding collisions with pedestrians. The path planning algorithm then extracts the optimal path from the constructed tree, containing all the feasible paths for the vehicle; a time law will be defined along the path, so to obtain a trajectory for the autonomous vehicle, that considers the actuation constraints, so to determine a feasible trajectory.

3.1.1 Path Planning problem formulation

The aim of path planning is to compute the desired path for the vehicle to move from a start to a goal configuration, provided as a sequence of waypoints in the environment. In this work, the goal is indeed to find the shortest path for the vehicle moving in a dynamic environment, avoiding collisions with moving obstacles (i.e. the pedestrians). Moreover, during the navigation, a re-planning of the shortest path may be required, due to changes in the dynamic environment.

To formalize the problem, some definitions are now presented:

- *State Space*: the state of the robot can be defined as $x = (q, t) \in X$, with $q = (x_p, y_p) \in Q$ being the single robot configuration in a 2D Cartesian global reference frame, and $Q \subset \mathbb{R}^{n_q}$ is the set of configurations of a robot with n_q independent variables. Being then X the state space of the robot and Q_{free} the set of obstacle-free robot configurations, it is possible to define the subset $X_{free} = \{x(q, t) \in X \mid q \in Q_{free}\}$, where X_{free} is the set of robot obstacle-free states;
- *Obstacle space*: the obstacle space $X_{obs} \subset X$ is the space of occupied robot states, such that $X_{free} = X/X_{obs}$;
- *Initial and final states*: considering a feasible path planning problem $(X_{free}, x_{init}, x_{goal})$, $x_{init} \in X_{free}$ is the initial robot state and $x_{goal} \subset X_{free}$ is the goal state to be reached.

It is now possible to define a feasible path for the vehicle as $\pi : [0, 1] \rightarrow X_{free}$ such that $\pi(0) = x_{init}$ and $\pi(1) = x_{goal}$.
Given then a feasible path $\pi(x_{start}, x_{goal})$, with x_{start}, x_{goal} corresponding to the start and goal states, and defining the path length as:

$$d_\pi(x_{start}, x_{goal}) = \sum_{i=1}^{N} \sqrt{(x_{P_{i_x}} - x_{P_{i-1_x}})^2 + (y_{P_{i_x}} - y_{P_{i-1_x}})^2} \tag{3.1}$$

where N is the number of points in the path, x_{P_i} is the generic point in $\pi(x_{start}, x_{goal})$ so that $x_{P_i} = (x_{P_{i_x}}, x_{P_{i-1_x}})$ are the robot spatial coordinates, it is possible to define the optimal path for the robot, denoted as $\pi^*(x_{start}, x_{goal})$ as the path with minimum length between the initial and goal states that does not collide with obstacles in the environment:

$$d_{\pi^*}(x_{start}, x_{goal}) = \underset{\pi \subset X_{free}}{\arg\min}\ d_\pi(x_{start}, x_{goal}) \tag{3.2}$$

Since in the dynamic environment the position of obstacles might change, the robot path may be modified during navigation. Hence, the optimal re-planning problem consists of finding π^* that satisfies the previous condition, for any time instant in which a re-planning is required.
More specifically, letting $\Pi(x_{curr})$ be the set of available collision-free path at a specific time instant, starting at the current robot state x_{curr} and ending in x_{goal}, the optimal path is the one that satisfies:

$$d_{\pi^*}(x_{curr}, x_{goal}) = \underset{\pi \subset \Pi(x_{curr})}{\arg\min}\ d_\pi(x_{start}, x_{goal}) \tag{3.3}$$

In the context of path planning, the RRT algorithm presented in [3] provides a probabilistically complete solution for the path planning problem: if a solution exists, the probability of finding a solution leans to one as the number of sampled states increases. The algorithm also allows to satisfy the obstacle avoidance

constraint (on fixed obstacles already in the map), by planning a path in the obstacle free state space. Various modified versions of RRT exist, such as the RRT* algorithm, that introduces the idea that asymptotic optimality can be guaranteed if new nodes, upon insertion, are allowed to "rewire" graph edges within their local neighbor nodes.
Anyhow, these algorithms lose in efficiency when the robot has to move in dynamic environments, in which the motion of the obstacles in the space requires to modify the path as the change of some obstacles position is detected. In this thesis, the RRTX algorithm was adopted, since when used in dynamic environments it has faster convergence rate that RRT*, keeping the asymptotic optimality property.

3.1.2 RRTX algorithm

RRTX is an asymptoticaly optimal sampling-based algorithm, that constructs a directed graph $G = (V, E)$, composed of nodes $v \in V$ randomly sampled from the robot free space X_{free} and edges $e \in E$, that represent the connections between two graph nodes, i.e. $e = (v, u)$, with $v, u \in V$. The edge connection is denoted as $\pi(v, u)$ in the algorithm description, and its length (i.e. its cost) is computed as the Eucledian distance between v and u, and is denoted with $d_\pi(v, u)$.
A tree T is then constructed from the graph G, that is defined as $T = (V_T, E_T)$, where $V_T \subset V$ and $E_T \subset E$, and is used to define a subtree, representing the optimal path to the goal vehicle state. Moreover, the tree T is rooted at the goal node v_{goal}, so that whenever an obstacle in the environment might cause a collision, only the terminal part of the tree has to be modified.
In order to express the edge connections between the nodes of G, RRTX associates different neighbor sets to each node. Given a node v in the graph, its neighbors are searched inside a range determined by a sphere centered in v, and whose radius is defined as follows:

$$r(|V|) = min(\frac{\gamma log|V|}{|V|}, \delta) \tag{3.4}$$

where $|V|$ is the number of elements of V and δ is the maximum allowed radius. It can be seen from 3.4 that the shrinking sphere radius r decreases as the size of the node set V increases. Thus, whenever a new node is extracted from X_{free}, the number of available neighbors decreases consequently.
Due to the directionality of the edged in the graph, the following sets are defined:

- *incoming neighbor set* $N^-(v)$, obtained as the set of all the nodes u connected to v through an edge ending in v;
- *outgoing neighbor set* $N^+(v)$, constituted of nodes u connected to v through an edge starting from v.

Starting from the definitions above, considering the set of original neighbors for v, $N_O(v)$, i.e. the neighbors of node v calculated at its insertion in the graph,

and the set of running neighbors $N_r(v)$ that contains the the neighbors of v inside the sphere having the current shrinking radius r, it is possible to define the two neighbors sets as:

$$\begin{aligned} N^-(v) &= N_O^-(v) \cup N_r^-(v) \\ N^+(v) &= N_O^+(v) \cup N_r^+(v) \end{aligned} \tag{3.5}$$

In equations 3.5, $N_O^-(v)$ and $N_O^+(v)$ are the set of incoming and outgoing original neighbors of v, while $N_r^-(v)$ and $N_r^+(v)$ are the set of incoming and outgoing running neighbors of v. Moreover, nodes always remember the original neighbors to which they are connected.
Two other set of nodes have to be defined, for each node v;

- the *child set* $\mathcal{C}_T^-(v)$: set of the incoming neighbors of v that are in the tree T.
- the *parent set* $p_T^+(v)$: set of the outgoing neighbors of v that are in the tree T. As the tree is rooted in v_{goal}, nodes in the parent set are closer to the goal node.

RRTX associates two costs to each node:

- $lmc(v) = \min_{u \in N^+(v)} (d_\pi(v,u) + lmc(u))$, is the minimum cost of reaching x_{goal} from v through G.
- $g(v)$ is the minimum cost of reaching x_{goal} from v thorugh the tree T

These two costs are used to determine the $\varepsilon - consistent$ nodes, that must satisfy the following condition:

$$g(v) - lmc(v) < \varepsilon \tag{3.6}$$

In this way, only $\varepsilon - consistent$ nodes are included in the subtree representing the optimal path. During the update of the graph, a node v becomes $\varepsilon - inconsistent$ whenever it is in collision with an obstacle. In such case, it is placed in a priority queue Q, that determines the order by which nodes can be extracted and added again to the tree, through a rewiring operation, if they become again $\varepsilon - consistent$ as a consequence of a variation in the obstacle set.
Moreover, nodes in the priority queue Q are ordered according to:

$$key\ value = min(lmc(v), g(v)) \tag{3.7}$$

The procedure of rewiring previously disconnected nodes to T is called *rewiring cascade*. The nodes that have been disconnected from T are called *orphan nodes* and are placed in the orphan node set V_T^C.
Then, as described in the previous chapter, if the following condition is satisfied:

$$\varepsilon_{tr} > \varepsilon_{max} \tag{3.8}$$

where ε_{max} is the maximum value of the tracking error, above which the re-planning of the trajectory is required, then a new optimal path is computed

between the node considered as the current robot node and the goal node. Moreover, the value for ε_{max} should be properly selected, so to enable the replanning action only in a condition for which the vehicle cannot recover the previously planned trajectory.

Algorithm structure

The structure of the RRTX algorithm, presented in [10] and [7], is now described, starting from the main code that describes the construction and update of the graph G and the tree T, until the robot reaches the goal.

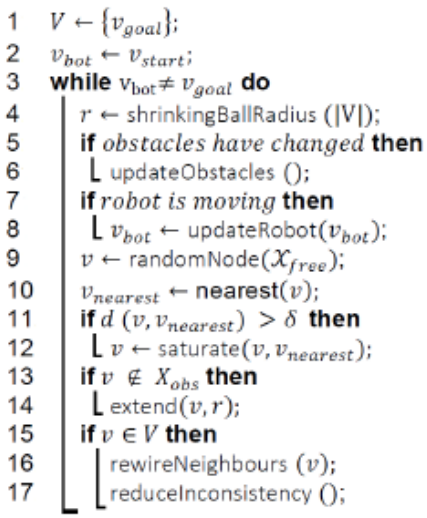


Figure 3.1: RRTX Main Code

When the algorithm starts, as shown in 3.1, graph G is initially constructed using only the goal and the start node, that are the goal and start robot locations. The node occupied by the robot at any time instant is denoted by v_{bot}; as the robot position at any time instant might not coincide with a node in G, v_{bot} is indeed the node in the graph closer to the current robot position. At each iteration, the algorithm uses the following functions:

- *shrinkingBallRadius*: function that defines the radius r of the ball inside which the algorithm searches for a neighbor node;
- *updateRobot*: function that updates the robot location in the graph, by updating the node v_{bot} according to the vehicle position at the considered time instant;
- *randomNode*: function that randomly samples a new node v from the free space X_{free};

- *nearest*: function that finds the node with the lower Eucledian distance from v, called $v_{nearest}$, among the nodes already in the graph;
- *saturate*: if the distance between v and $v_{nearest}$ is higher than the predefined maximum distance δ, this function creates a new node called $v_{saturated}$ at distance δ from v, along the line connecting v and $v_{nearest}$;
- *extend*: as shown in figure 3.2, this function tries to find a parent node for v, using the function $findParent$. If a parent node for v is found, the *extend* function adds v to the set of nodes V, and to the child set of the parent of v. The neighbor sets of v and of its neighbors is the updated accordingly;
- $findParent$: given the set of neighbors of v that are inside the shrinking sphere, a node u that makes the $lmc(v)$ cost the lower is selected as parent of v, provided that the edge (v, u) is not in collision with any obstacle;

```
1  V_near ← near(v, r)
2  findParent(v, V_near)
3  if p_T^+(v) = ∅ then
4      return
5  V ← V ∪ {v}
6  C_T^-(p_T^+(v)) ← C_T^-(p_T^+(v)) ∪ {v}
7  forall u ∈ V_near do
8      if π(v, u) ∩ X_obs = ∅ then
9          N_0^+(v) ← N_0^+(v) ∪ {u}
10         N_r^-(u) ← N_r^-(u) ∪ {v}
11     if π(u, v) ∩ X_obs = ∅ then
12         N_r^+(u) ← N_r^+(u) ∪ {v}
13         N_0^-(v) ← N_0^-(v) ∪ {u}
```

Figure 3.2: RRTX extend function

```
1 forall u ∈ U do
2     π(v, u) ← computeTrajectory(X, v, u)
3     if d_π(v, u) ≤ r and
      lmc(v) > d_π(v, u) + lmc(u) and π(v, u) ≠ ∅
      and X_obs ∩ π(v, u) = ∅ then
4         p_T^+(v) ← u
5         lmc(v) ← d_π(v, u) + lmc(u)
```

Figure 3.3: RRTX findParent function

- *rewireNeighbors*: if v has been added to the node set V, this function

(figure 3.4) tries to set v as a parent of its incoming neighbors. Through the *cullNeighbors* function shown in figure 3.5, all the outgoing running neighbor nodes of v, $u \in N_r^+(v)$ outside the sphere centered in v with radius r and that are not part of T are deleted. Then, given all the incoming neighbors of v, $u \in N^-(v)$, that are not parents of v, the function sets v as a parent of u, provided that the $lmc(u)$ cost is lower. Moreover, if u is ε-inconsistent, it is placed in the priority queue Q, through the $verifyQueue$ function;

```
1 if g(v) − lmc(v) > ε then
2     cullNeighbors(v, r)
3     forall u ∈ N⁻(v) \ {p_T^±(v)} do
4         if lmc(u) > d_π(u, v) + lmc(v) then
5             lmc(u) ← d_π(u, v) + lmc(v)
6             makeParentOf(v, u)
7             if g(u) − lmc(u) > ε then
8                 verrifyQueue(u)
```

Figure 3.4: RRTX rewireNeighbors function

```
1 forall u ∈ N_r^+(v) do
2     if r < d_π(v, u) and p_T^±(v) ≠ u then
3         N_r^+(v) ← N_r^+(v) \ {u}
4         N_r^−(u) ← N_r^−(u) \ {v}
```

Figure 3.5: RRTX cullNeighbors function

- *reduceInconsistency*: this function performs the rewiring cascade operation. If the node v_{bot} is $\varepsilon - inconsistent$, or the node with lower key value in Q has a key value lower than v_{bot}, such node is rewired in the tree T using the *rewireNighbors* function previously described, and by setting its $g(v)$ cost equal to its $lmc(v)$ cost it becomes $\varepsilon - consistent$, as shown in figure 3.6;

```
1 while size(Q) > 0 and
  (keyLess(top(Q), v_bot) or lmc(v_bot) ≠ g(v_bot)
   or g(v_bot) = ∞ or Q ∋ v_bot) do
2     v ← pop(Q)
3     if g(v) − lmc(v) > ε then
4         updateLMC(v)
5         rewireNeighbors(v)
6     g(v) ← lmc(v)
```

Figure 3.6: RRTX reduceInconsistency function

- *updateObstacles*: this function, shown in figure 3.7, allows to updates the obstacle description in the algorithm using two main functions: the *removeObstacle* function (figure 3.8) and the *addNewObstalce* function (figure 3.9);
- *removeObstacle*: this function removes an obstacle that disappears from the environment. Firstly, it recalculates the distance d_π of the edges (v, u) that were in collision with only that obstacle, and using the new d_π value it recalculates the $lmc(v)$ of node v from the edge (v, u). The computation of the new d_π will be discussed in the following section;
- *addNewObstalce*: function that adds a new obstacle to the obstacle set. The distance of each edge in collision with the new obstacle, $d_\pi(v, u)$ is set to infinite. All the staring node v of these edges already in the tree T are placed in the orphan node set V_T^C. Then, the *propagateDescendants* function, shown in figure 3.10 propagates the effect of the added obstacle among all the nods in T. For each node v of the orphan set and for each children node of v, the g cost of each outgoing neighbor and parent of those nodes is set to infinite. Then, the orphan set is emptied, setting to infinite the g and lmc cost of its nodes and removing them from T.

```
1  if ∃O : O ∈ 𝒪 ∧ O has vanished then
2  |   forall O : O ∈ 𝒪 ∧ O has vanished do
3  |   |   removeObstacle(O)
4  |   reduceInconsistency()
5  if ∃O : O ∉ 𝒪 ∧ O has appeared then
6  |   forall O : O ∉ 𝒪 ∧ O has appeared do
7  |   |   addNewObstacle(O)
8  |   propogateDescendants()
9  |   verrifyQueue(v_bot)
10 |   reduceInconsistency()
```

Figure 3.7: RRTX updateObstacles function

```
1  E_O ← {(v, u) ∈ E : π(v, u) ∩ O ≠ ∅}
2  𝒪 ← 𝒪 \ {O}
3  E_O ← E_O \ {(v, u) ∈ E : π(v, u) ∩ O' ≠ ∅
   for some O' ∈ 𝒪}
4  V_O ← {v : (v, u) ∈ E_O}
5  forall v ∈ V_O do
6  |   forall u : (v, u) ∈ E_O do
7  |   |   d_π(v, u) ← recalculate d_π(v, u)
8  |   updateLMC(v)
9  |   if lmc(v) ≠ g(v) then verrifyQueue(v)
```

Figure 3.8: RRTX removeObstacle function

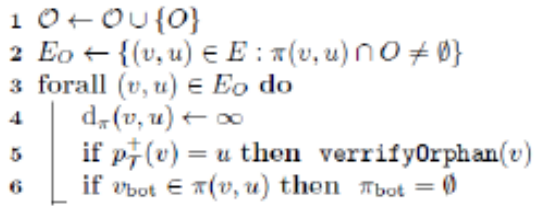


Figure 3.9: RRTX addNewObstacle function

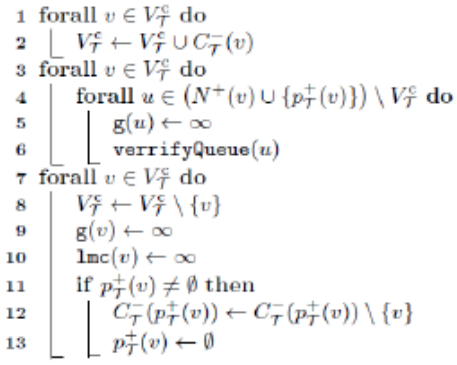


Figure 3.10: RRTX propagateDescendants function

3.1.3 RRTX with pedestrian motion prediction

Given the future prediction of obstacles position it is possible to obtain, for each of the currently detected obstacles, a prediction area describing the space that will be occupied by the obstacle. In this work, the motion of obstacles is assumed as a linear displacement with constant velocity inside the described time window. In figure 3.11 an example of the prediction area is shown, where the blue circle represents the obstacle at its current position, while the rectangle describes the future predicted area that will be occupied by the pedestrian.

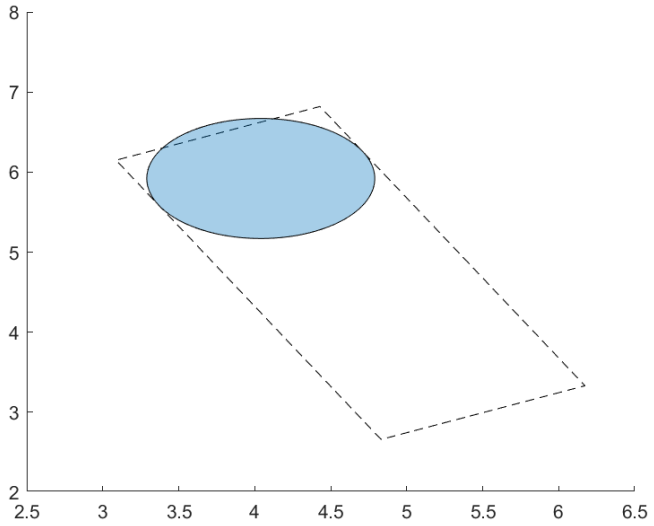


Figure 3.11: Pedestrian future predicted occupied area

The shape of the predicted area is hence obtained considering the size of the obstacle, and propagating such object along its future predicted positions. The obstacle size also accounts for the size of the wheelchair, so that the shape of the robot is considered in the evaluation of the collision between the edges and the future predicted area.
To properly include the knowledge of the future prediction area of each detected pedestrian, the graph generated by the algorithm has to be modified accordingly. Whenever an edge $e(v, u)$ of the graph is totally or partially inside the future predicted area of a pedestrian, its cost is modified (i.e. the length of the edge). To modify the cost of an edge involved in a collision condition, the intersection point between such edge and the considered pedestrian prediction area is computed. As more than one intersection point can be found, the one closer to the current pedestrian position is considered. Then, the distance between the obtained intersection point and the current pedestrian position is considered to modify the edge cost in the graph. More specifically, given the distance d_{pr} between the pedestrian and the selected intersection point, denoting a potential collision, the edge cost can be modified as follows:

$$d_\pi(v, u) = d_\pi(v, u)(1 + \frac{1}{d_{pr}}) \tag{3.9}$$

so that an intersection point closer to the pedestrian represent an higher collision danger, allowing to properly modify the vehicle path during the navigation.

3.2 Trajectory generation

Once the optimal collision-free path is obtained as a sequence of waypoints, a properly defined time law can be introduced, so to interpolate motion between the intermediate points, and provide a trajectory for the vehicle. In this work, a Trapezoidal Velocity Profile (TVP) has been selected as the time law to be imposed on the obtained path. TVP is a velocity profile composed of three different sections:

- *constant acceleration section*
- *constant velocity*
- *constant deceleration section*

The implementation presented allows to generate a trajectory as a sequence of points, according to the controller sampling time τ, in the free space, connecting the path waypoints. Moreover, this implementation allows to define constraints in the velocity and acceleration, so to obtain a trajectory that satisfies the constraints on maximum performances of the vehicle.
The implementation of TVP then return position, velocity and acceleration profile, as in figure 3.12, describing the trajectory planned from start to goal position.

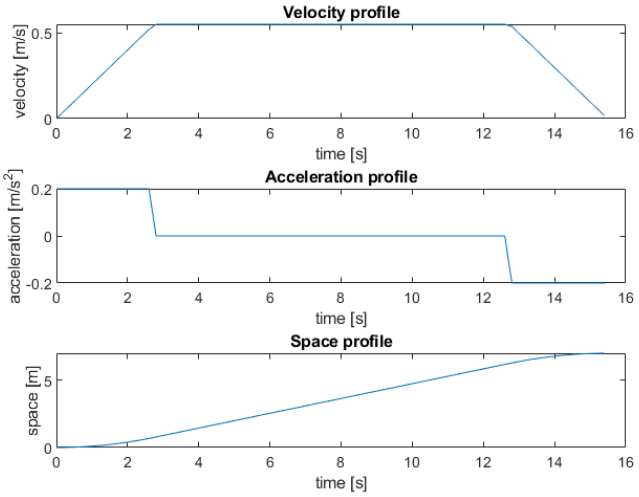


Figure 3.12: Position, velocity and acceleration profile for the generated trajectory

Since the path obtained from path planning is composed of a sequence of points, the overall trajectory from the start position to the goal position is

obtained through the interpolation of the path waypoints, and then applying the velocity profile to the full interpolated path. Moreover, when a replanning action is required, the new trajectory will be computed considering the current velocity of the vehicle, so to avoid the generation of a trajectory starting from a zero initial velcity, while the vehicle is moving with a velocity higher than $0[m/s]$

Chapter 4

Robot and Pedestrian model

This chapter starts with a description of the robot model, together with the model of the moving obstacles, i.e. the pedestrians, with which the vehicle must interact during its navigation in the environment.
In the first part, two kinematic models for the vehicle are introduced, together with the description of the similarities between them. The selected non-linear model will be then linearized through the *feedback linearization* technique, so to obtain a model suitable for a Model Predictive Control (MPC) design.
Then, a model that describes the pedestrian is presented, that is necessary for the robot to predict the motion of every moving human, together with a description of its *Personal Space*, which will be described in the relative section.

4.1 Autonomous vehicle

Making reference to [8] and [11], the vehicle considered in this work is a motorized wheelchair for indoor and outdoor use, the Degonda Twist t4 2x2, shown in Figure 4.1, produced by Degonda Rehab SA. This wheelchair is equipped with two rear driving wheels, whose motors are characterized by a maximum power of 0.35 [kW], and three caster wheels with stabilization function.
To detect the relative distance with respect to surrounding objects, the vehicle is endowed with two time-of-flight laser sensors SICK TiM561, characterized by a maximum angular amplitude of 270° with 0.33° of resolution, a maximum range of 10 [m] and a scanning frequency of 15 [Hz]. To guarantee a complete vision of the environment, the two sensors are installed in opposite edges, as shown in Figure 4.2

Figure 4.1: Wheelchair Degonda Twist t4 2x2

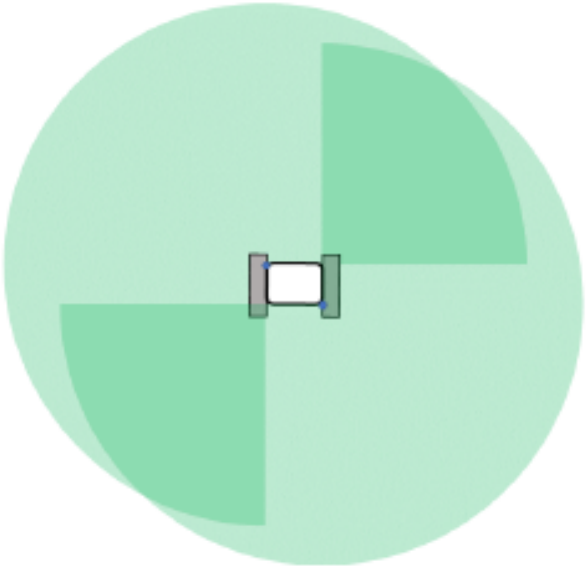


Figure 4.2: Angular rate

4.1.1 Kinematic model

The motion of the considered autonomous vehicle can be represented, from a kinematic point of view, using a differential drive robot model. A differential drive is a robot with two independently controlled wheels with a common axis of rotation, and one or more passive wheels to balance the robot.
The two fixed wheels are actuated by two independent motors, having rotational velocity ω_R and ω_L. Thus the linear velocities are $v_R = \omega_R R$ and $v_L = \omega_L R$, where R is the radius of the wheels. The vehicle control variables are its linear

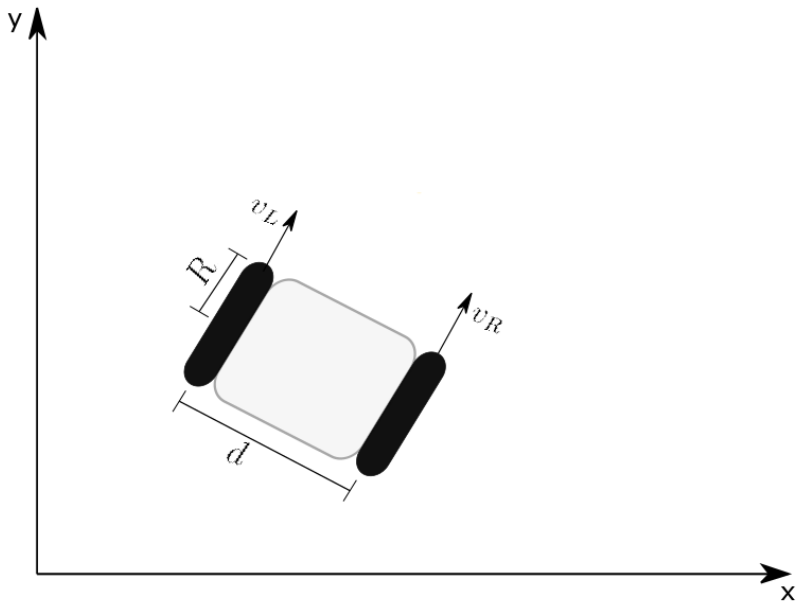


Figure 4.3: Differential drive robot

and angular velocity, v and ω respectively. Through simple kinematic considerations, it is possible to relate them to the rotational velocities of the two wheels:

$$\begin{aligned} v(t) &= R\frac{\omega_R(t) + \omega_L(t)}{2} \\ \omega(t) &= R\frac{\omega_R(t) - \omega_L(t)}{d} \end{aligned} \tag{4.1}$$

where d is the robot baseline.
From a kinematic point of view, this vehicle model is equivalent to a unicycle

model (Figure 4.4), and can be described by the following nonlinear equations:

$$\begin{cases} \dot{x}(t) &= v(t)\cos\theta(t) \\ \dot{y}(t) &= v(t)\sin\theta(t) \\ \dot{\theta}(t) &= \omega(t) \end{cases} \tag{4.2}$$

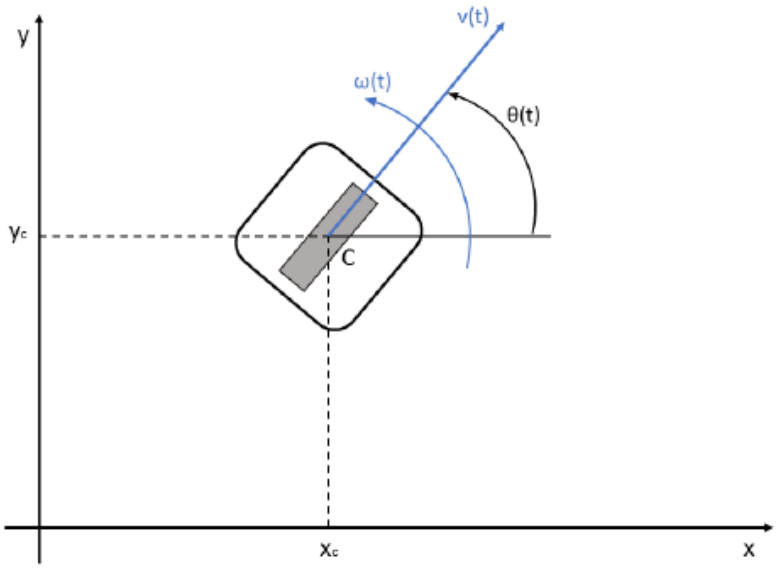


Figure 4.4: Unicycle robot

In this set of nonlinear equations, $x(t)$, $y(t)$ and $\theta(t)$ are the state variables of the model, representing the wheel contact point and the orientation of the robot in a global reference frame, while $v(t)$ and $\omega(t)$ are the input variables. Moreover, to simulate the uncertainty on the vehicle localization in a real navigation scenario, it is possible to include a noise component in the three kinematic states in equation 4.2. As the estimate on θ is usually the one affected by an higher uncertainty, the θ dynamics has been modified adding a noise source, modeled as a normally distributed random variable: $w_\theta = \mathcal{N}(\mu, \sigma^2)$

4.1.2 Feedback Linearization

The previously described model characterized by a nonlinear relationship between the state variables ($x(t)$, $y(t)$, $\theta(t)$) and the control variables ($v(t)$ and $\omega(t)$) is not suitable for the development of a linear controller. However, thanks to an inner feedback-linearizing control loop described in [4], it is possible to approximate the system as a particle, whose dynamics can be described by means

of a linear relationship.
This approach is based on a coordinates transformation, obtained considering a point P placed at distance ε from the unicycle wheel axle, along the longitudinal direction (Figure 4.5). Coordinates of point P can be now expressed, with respect to the global reference system, as:

$$\begin{cases} x_P(t) = x_C(t) + \varepsilon \cos\theta(t) \\ y_P(t) = y_C(t) + \varepsilon \sin\theta(t) \end{cases} \tag{4.3}$$

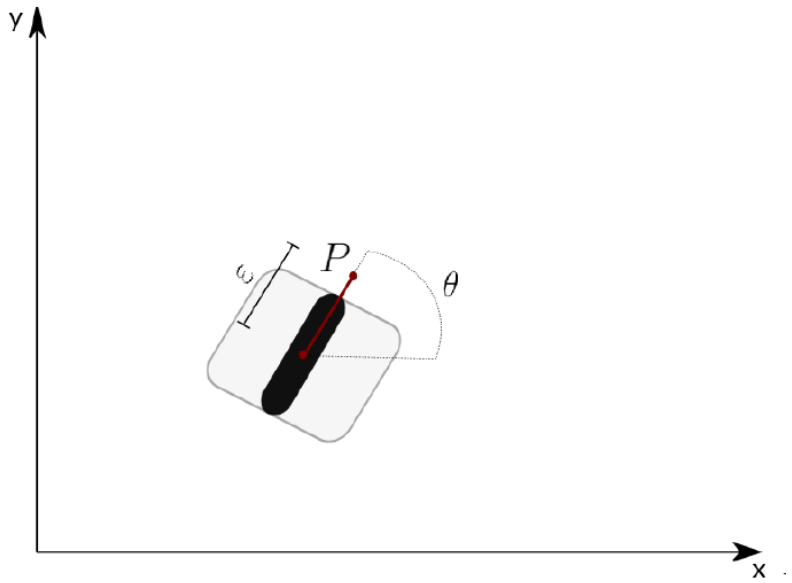


Figure 4.5: Feedback linearization for the unicycle model, considering a point P at distance ε

where $x_C(t)$, $y_C(t)$ and $\theta(t)$ are the position and orientation of the vehicle, with respect to a global reference frame.
By deriving with respect to time:

$$\begin{cases} \dot{x}_P(t) = \dot{x}_C(t) - \varepsilon \sin\theta(t)\dot{\theta}(t) \\ \dot{y}_P(t) = \dot{y}_C(t) + \varepsilon \cos\theta(t)\dot{\theta}(t) \end{cases} \tag{4.4}$$

Then, recalling that $\dot{x}_C(t) = v(t)\cos\theta(t)$, $\dot{y}_C(t) = v(t)\sin\theta(t)$ and $\dot{\theta}(t) = \omega(t)$, it is possible to rewrite the previous relationship in matrix form:

$$\begin{bmatrix} \dot{x}_P(t) \\ \dot{y}_P(t) \end{bmatrix} = \begin{bmatrix} v_{P_x}(t) \\ v_{P_y}(t) \end{bmatrix} = \begin{bmatrix} \cos\theta(t) & -\varepsilon\sin\theta(t) \\ \sin\theta(t) & \varepsilon\cos\theta(t) \end{bmatrix} \begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} \tag{4.5}$$

The transformation matrix for feedback linearization can then be defined as:

$$T(\theta, \varepsilon) = \begin{bmatrix} \cos\theta(t) & -\varepsilon \sin\theta(t) \\ \sin\theta(t) & \varepsilon \cos\theta(t) \end{bmatrix} \tag{4.6}$$

Matrix $T(\theta, \varepsilon)$ is non-singular, and thus invertible, $\forall\theta$ and $\forall\varepsilon \neq 0$, and it is then possible to obtain a relationship between the variables of the linearized system ($v_{P_x}(t)$ and $v_{P_y}(t)$) and the original one ($v(t)$ and $\omega(t)$).
The final model obtained from the feedback linearization procedure is described by a linear decoupled system, characterized by two integrators, as shown in Figure 4.6:

$$\begin{cases} \dot{x}_P(t) &= v_{P_x}(t) \\ \dot{y}_P(t) &= v_{P_y}(t) \end{cases} \tag{4.7}$$

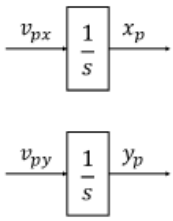


Figure 4.6: Double integrator system

However, the previous system describes a continuous time model. To make it compatible with the discrete time nature of the MPC, it has to be discretized. In this work, the forward Euler method is used, which consists in imposing:

$$s = \frac{z-1}{\tau} \tag{4.8}$$

where τ is the sampling time considered for the controller, obtaining the following discrete model of the system:

$$\begin{cases} x_P(k+1) = x_P(k) + \tau v_{P_x}(k) \\ y_P(k+1) = y_P(k) + \tau v_{P_y}(k) \end{cases} \tag{4.9}$$

which can be rewritten in compact form as:

$$\xi(k+1) = A\xi(k) + Bu(k) \tag{4.10}$$

where:

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \; B = \begin{bmatrix} \tau & 0 \\ 0 & \tau \end{bmatrix} \; \xi(k) = \begin{bmatrix} x_P(k) \\ y_P(k) \end{bmatrix} \; u(k) = \begin{bmatrix} v_{P_x}(k) \\ v_{P_y}(k) \end{bmatrix} \tag{4.11}$$

It is also possible to show that, if the point P is forced to move along a straight line, the wheelchair automatically aligns itself to the motion of point P, excluding from the system dynamics unnatural movements of the vehicle, such as backward motion. As described in [8], by imposing as inputs for the linearized system $v_{P_x} = v_p cos\theta_p$ and $v_{P_y} = v_P sin\theta_p$, with $v_P > 0$ and θ_p constant, the wheelchair is forced to perform a uniform motion along a straight line, and the dynamics of the closed-loop system can be rewritten as:

$$\begin{cases} \dot{x(t)} = v_p(\cos\theta(t)\cos\theta_p + \sin\theta(t)\sin\theta_p)\cos\theta(t) \\ \dot{y(t)} = v_p(\cos\theta(t)\cos\theta_p + \sin\theta(t)\sin\theta_p)\sin\theta(t) \\ \dot{\theta(t)} = -\frac{v_p}{\varepsilon}\sin(\theta(t) - \theta_p) \end{cases} \tag{4.12}$$

Being θ and hidden state in the final linearized system previously described, it means it is not controllable. It is hence crucial to ensure that its dynamics is asymptotically stable.
Defining then $\Delta\theta(t) = \theta(t) - \theta_p$, the equation of heading dynamics becomes:

$$\dot{\theta(t)} = -\frac{v_p}{\varepsilon}\sin\Delta\theta(t) \tag{4.13}$$

and from the obtained equation it is possible to conclude that:

- the equilibrium conditions $\theta = \theta_p + 2k\pi$, with $k = 1, 2, \ldots$, are asymptotically stable, with a basin of attraction defined by the open interval $(\theta_p - (2k-1)\pi, \theta_p + (2k+1)\pi)$;
- the equilibrium conditions $\theta = \theta_p + (2k+1)\pi$, with $k = 1, 2, \ldots$ are unstable

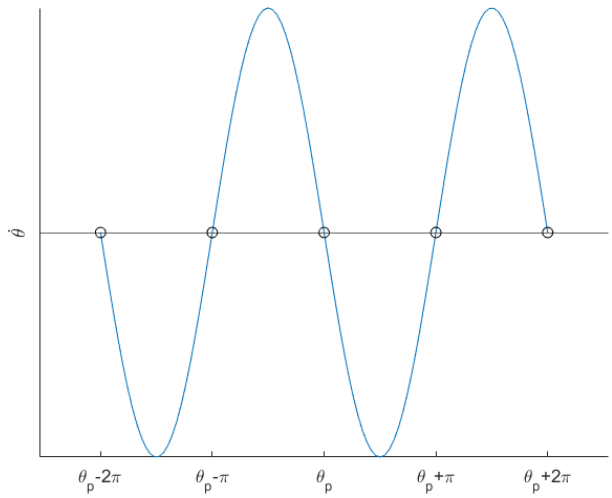


Figure 4.7: $\dot{\theta}$ dynamics with asymptotically stable and unstable equilibria, identified by the black circles

This property obtained for the θ dynamics, and in particular the definition of the unstable equilibrium conditions, show that the wheelchair automatically aligns itself to the motion of point P, when forced to follow a uniform motion along a straight line, practically excluding unnatural movements of the vehicle, such as backward motion.

4.2 Pedestrian model

In the context of local planning with obstacle avoidance, each pedestrian in the environment has to be characterized as a moving obstacle. Moreover, the robot should behave in a way that is accepted by humans in the environment, so to guarantee the human comfort in the interaction with the wheelchair. To this aim, the concept of *Perosnal Space function* of a pedestrian and the *Virtual Box* are introduced.

4.2.1 Kinematic model

The main assumption needed to apply the obstacle avoidance algorithm is that the controller will be able to predict the vehicle motion, within a short time window, exploiting a specific motion model for the moving obstacles in the environment. In particular, the hypothesis adopted in this thesis for the motion of pedestrians (moving obstacles) is that, within a short time window, their velocity will be constant.
Human motion can be hence modeled as a rectilinear motion of a particle moving on the $x - y$ plane, with constant velocity:

$$\begin{cases} \dot{x}_{ped}(t) = v_{ped_x} \\ \dot{y}_{ped}(t) = v_{ped_y} \end{cases} \tag{4.14}$$

where v_{ped_x} and v_{ped_y} are the components of pedestrian velocity. Following the same procedure described for the wheelchair kinematic model. the pedestrian motion model is discretized, so to obtain:

$$\begin{cases} x_{ped}(k+1) = x_{ped}(k) + \tau v_{ped_x} \\ y_{ped}(k+1) = y_{ped}(k) + \tau v_{ped_y} \end{cases} \tag{4.15}$$

where τ is the sampling time of the controller. This motion model will be then used to predict the motion of the pedestrian, knowing its initial position (x_{ped_0}, y_{ped_0}) and velocity, that can be obtained from sensors readings.

4.2.2 Virtual Box

Even if the motion of a pedestrian can be modeled as the linear motion of a particle in the plane, in the context of obstacle avoidance the size of the pedestrian is a relevant parameter for the definition of a region in which the vehicle

is free to move, without causing a collision. In order to account for the size of the pedestrian, the concept of *Virtual Box* is now introduced. In fact, the obstacles detected by the sensors are modeled as circular virtual boxes, so to ensure the admissibility of the convex optimization problem.
In the considered system, the perception of the surrounding space (and hence moving and static obstacles, including pedestrians) is performed through two time-of-flight laser sensors. As explained in previous works, these sensors produce a dense set of measurements, representing the view from the wheelchair perspective, that can be transformed into a *PointCloud* in the global reference frame, that is a solution more suitable for control purposes. For each obstacle, represented by a specific subset of points, the approximate center can be calculated, as well as the radius of a circle that inscribes all the related data points, which is then the obstacle virtual box, i.e. the space occupied by a pedestrian.

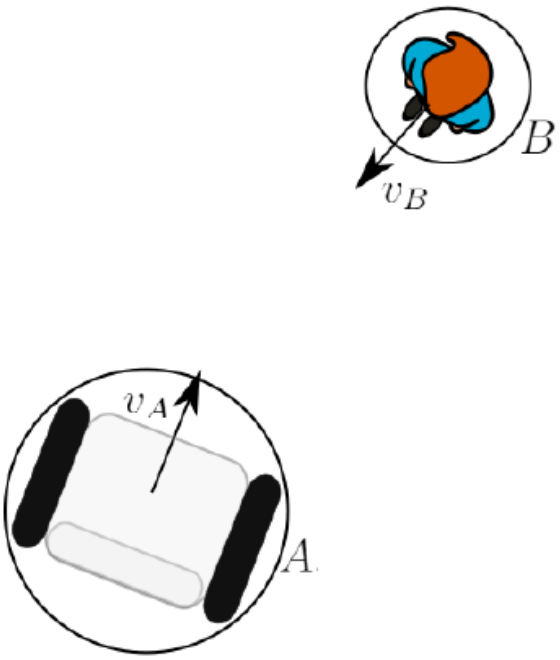


Figure 4.8: Vehicle and pedestrian representation, moving respectively with velocities v_A and v_B. The circle enclosing the vehicle represents its dimension.

To properly consider the wheelchair dimensions, the virtual box is enlarged by an amount equal to the radius of the circle in which the vehicle can be inscribed, as shown in Figure 4.8, so to map the obstacle in the vehicle configuration space.

The obtained final virtual box, containing the pedestrian, will be then used for the computation of the MPC state constraints in the form of linear inequalities dividing the state space into two half-planes, one of which will be the admissible region for the vehicle motion. This indeed allows to obtain a convex admissible region, as it will be described in the following sections.

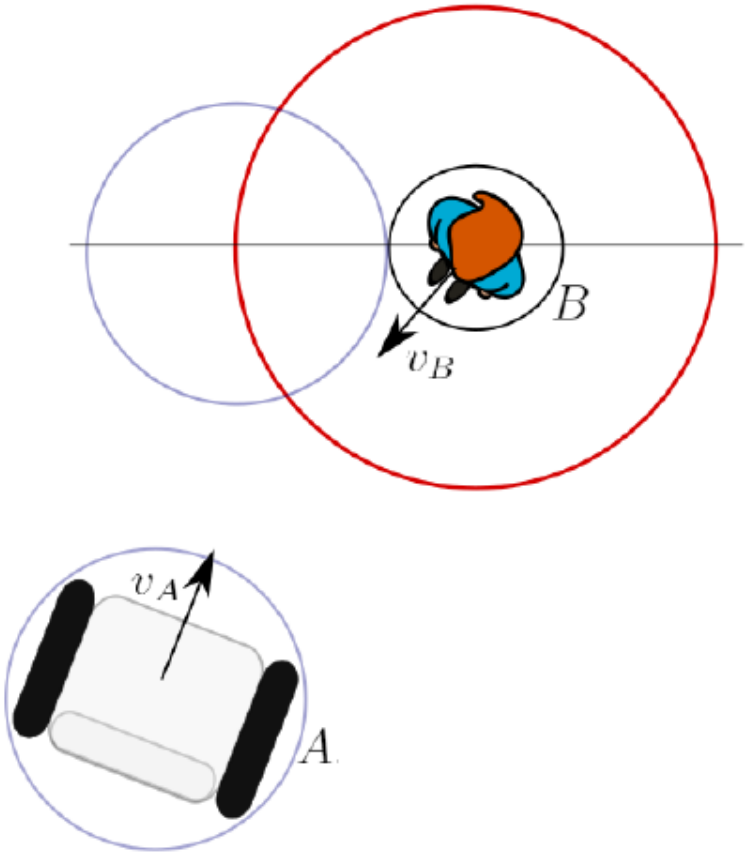


Figure 4.9: Pedestrian enlarged virtual box in red, accounting for the wheelchair dimension

4.2.3 Personal Space Function

A crucial aspect for the navigation task in crowded environment is the ability to detect and predict the human behaviour, so that the wheelchair can adapt its motion accordingly. In such context, guaranteeing a feasible and safe trajectory for the wheelchair is not enough. It is indeed required to perform a motion that

is perceived as safe from humans, respecting social conventions.
The definition of the *Personal Space function* (PSf) exploits the concept of *Proxemics* to determine a model for the description of the distance that the vehicle should try to keep from other people, that is socially accepted and perceived as safe and natural from pedestrians.
The concept of Proxemics was first introduced by Edward T. Hall [1], describing human personal space as the distance that each human try to keep from other people, according to subtle cultural rules. In his study, Hall divided the personal space of a human in 4 different circular areas, to differentiate between

- *Intimate space*
- *Personal space*
- *Social space*
- *Public space*

each including an incrementally larger region of space.
However, this result proved to be true only in static situations. More recent works extended Hall's personal space definition to include the effects of motion. For this reason the personal space for a moving person is modeled as two halves of a 2D Gaussian function, as described in [5] and [12], whose size is related to the pedestrian walking speed.
More specifically, considering the autonomous vehicle and the pedestrian moving towards each other, the size of the personal space function depends on the relative velocity between them: below the X-axis, the function is a symmetric Gaussian with variance:

$$\sigma_x = \sigma_y = 3v \tag{4.16}$$

while above the X-axis variances are:

$$\sigma_y = 1.5\sigma_x \tag{4.17}$$

where v is the relative velocity, as displayed in figure 4.10.
The personal space function can be then represented as the composition of two functions, an elliptical function in front of the person and a symmetrical function behind. By considering an angle $\alpha \in [-\pi, \pi]$, positive in counter clockwise direction:

$$PSf(x,y) = \begin{cases} \frac{(x-x_{ped})^2}{\sigma_x^2} + \frac{(y-y_{ped})^2}{\sigma_y^2} = 1 \quad , \quad 0 \leq \alpha \leq \pi \\ (x-x_{ped})^2 + (y-y_{ped})^2 = \sigma_x^2 \quad , \quad -\pi \leq \alpha \leq 0 \end{cases} \tag{4.18}$$

where (x_{ped}, y_{ped}) is the current position of the pedestrian, σ_x is the radius of the circle defining the personal space behind the pedestrian and σ_y describes the personal space in front of it. Considering instead the case of a fixed obstacle, i.e. a pedestrian that is not moving, its personal space is modeled as a circle, still depending on the relative velocity, that now is equal to the velocity of the

vehicle. Keeping the same notation, the personal space function for a fixed obstacle is a symmetric Gaussian function; hence, by setting $\sigma_x = \sigma_y = 3v$ both above and below the X-axis, the personal space function defined by equation (4.18) describes a circle.

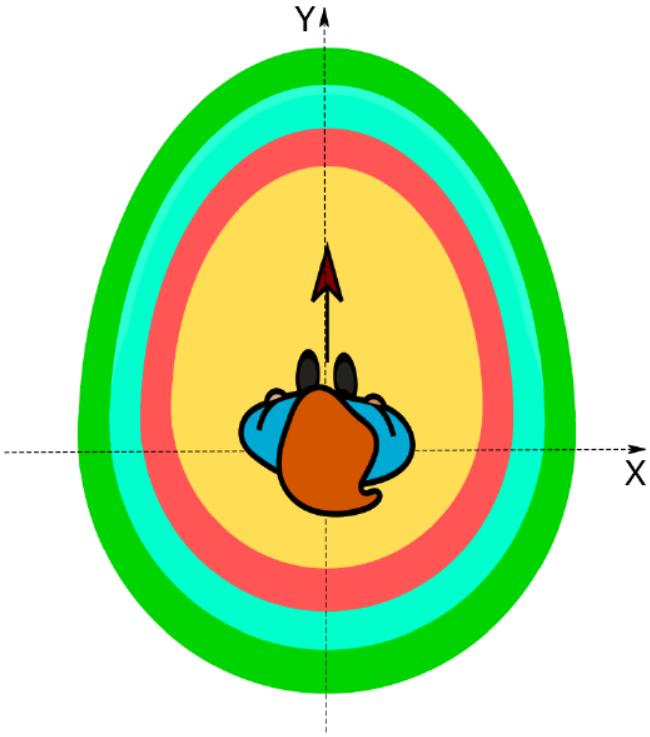


Figure 4.10: Personal space function for a person moving along the positive Y axis, with a relative velocity of 1 m/s

Chapter 5

Control Problem Formulation

The focus of this chapter is the formulation of the control problem. First, the obstacle avoidance strategy adopted in this thesis is presented, starting from the description of the collision detection methodology and the definition of the obstacle-free space for the vehicle. Then, these concepts will be integrated in the control problem for the definition of position constraints for a Model Predictive Control approach, as well as constraints on velocity and acceleration of the vehicle.

5.1 Obstacle Detection and Avoidance

In this work, state constraints defining the admissible region for the robot are selected based on the concept of *Velocity obstacles* [2], that allows to map the dynamic environment, i.e. the moving pedestrians, in the robot velocity space. This method allows to determine potential collisions, using information about the robot time-varying position and velocity in the environment, as well as the obstacles positions and velocities. By mapping the environment in the vehicle velocity space, it is possible to determine the *Collision Cone* of the vehicle. This allows to retrieve a region of input velocities that, if applied to the vehicle, will lead to a collision with an obstacle at some future time within a given prediction horizon.
In this section, the concept of Velocity Obstacle is presented for a single obstacle, but the approach can be easily extended to an arbitrary number of objects.
Since both the vehicle and the obstacle are inscribed in a circular virtual box, as previously described, two circular objects are now considered:

- A: circle representing the vehicle
- B: circle representing the obstacle

Moreover, the two objects are moving with velocities v_A, v_B at time t, respec-

tively.
To compute the Collision Cone, the obstacle B has to be mapped into the configuration space of the vehicle.This can be done reducing the vehicle representation to a point $\hat{A}$ and by enlarging B by the radius of circle A, that will be called $\hat{B}$, as shown in figure 5.1.

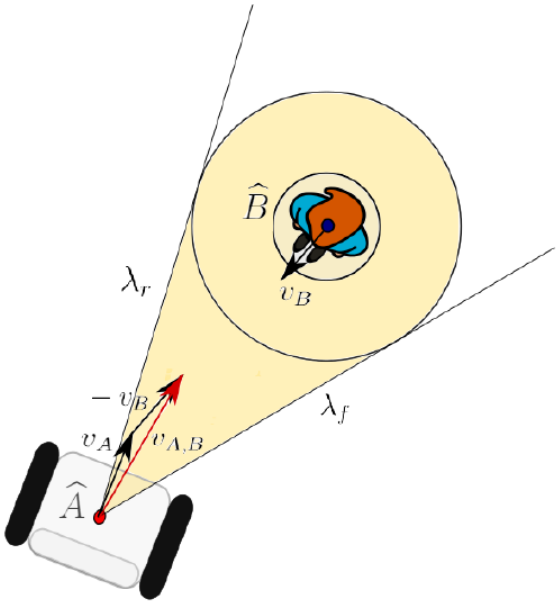


Figure 5.1: The yellow region represents the Collision Cone, the set of relative velocities that might lead to a collision. It is defined as the sector with apex in $\hat{A}$ and bounded by the two lines λ_r and λ_f

The state of each object can be then represented by its position and the velocity vector attached to its center.
The Collision Cone $CC_{A,B}$ is then defined as the set of colliding relative velocities between $\hat{A}$ and $\hat{B}$:

$$CC_{A,B} = \{v_{A,B} \mid \lambda_{A,B} \cap \hat{B} \neq 0\} \tag{5.1}$$

where $\lambda_{A,B}$ is the direction of $v_{A,B}$, that is the relative velocity between the two moving objects described:

$$v_{A,B} = v_A - v_B \tag{5.2}$$

The Collision Cone is determined as the sector having apex in $\hat{A}$ and bounded by the two lines tangent to $\hat{B}$, λ_r and λ_f. All the relative velocities lying between

those two lines, and hence inside the Collision Cone, are classified as velocities that will potentially lead to a collision with the moving obstacle, and hence have to be avoided.
Relative velocity vectors lying outside the Collision Cone are guaranteed to be collision-free, provided that the obstacle keeps the same shape and speed. Moreover, this collision-free condition, and hence the collision cone defined, is specific for each pedestrian-robot pair.

5.1.1 Collision Detection

The concept of Velocity Obstacles previously described is now applied considering the obstacle enlarged virtual box, so to determine the set of lines defining the convex obstacle-free region of space in which the robot can move. As introduced in the previous section, any relative velocity lying between the two lines tangent to the obstacle enlarged virtual box $\hat{B}$ and connecting the vehicle center $\hat{A}$, might lead to a collision; to avoid the collision between the robot and the moving obstacle, the vehicle free space must be properly limited.

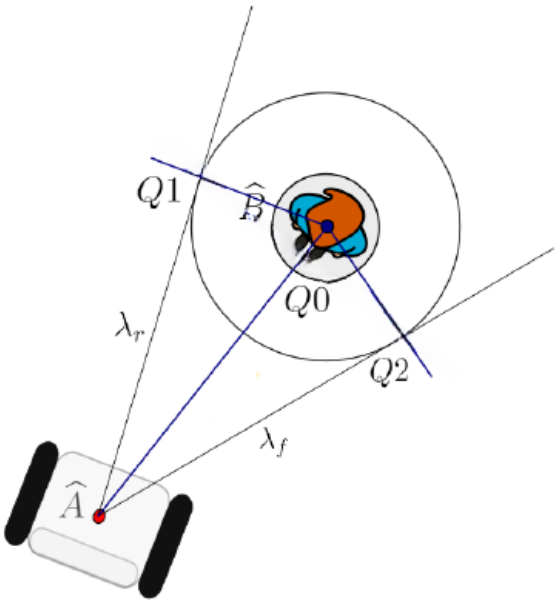


Figure 5.2: The two tangent points Q_1 and Q_2 on the obstacle virtual box, determined through trivial geometric considerations.

To determine whether the relative velocity between the vehicle and the pedestrian lies within the collision cone, variables ν_1 and ν_2 are introduced, corre-

sponding to directions of the two boundary tangent lines λ_f and λ_r, respectively:

$$\begin{aligned} \nu_1 &= \begin{bmatrix} Q_{1_x} - P_{r_x} \\ Q_{1_y} - P_{r_y} \end{bmatrix} \\ \nu_2 &= \begin{bmatrix} Q_{2_x} - P_{r_x} \\ Q_{2_y} - P_{r_y} \end{bmatrix} \end{aligned} \tag{5.3}$$

where Q_1 and Q_2 are the two tangent points on the obstacle virtual box, while P_r represents the robot position at the current time instant. It is then possible to compute the relative velocity as:

$$\nu_R = \nu_r - \nu_{obs} = \begin{bmatrix} \nu_{r_x} - \nu_{obs_x} \\ \nu_{r_y} - \nu_{obs_y} \end{bmatrix} \tag{5.4}$$

where ν_r and ν_{obs} are the vectors representing the robot and obstacle velocity directions.
The relative velocity vector ν_R can be then rewritten as a linear combination of ν_1 and ν_2, defining the Collision Cone:

$$\nu_R = \alpha_1 \nu_1 + \alpha_2 \nu_2 \tag{5.5}$$

where α_1 and α_2 are two constant values.
The values of these two constant values can be then used to analyse the position of the relative velocity vector, with respect to the Collision Cone. More specifically, rewriting the previous equation with respect to the global reference:

$$\begin{bmatrix} \nu_{R_x} \\ \nu_{R_y} \end{bmatrix} = \begin{bmatrix} \nu_{1_x} & \nu_{2_x} \\ \nu_{1_y} & \nu_{2_y} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} \tag{5.6}$$

$$\alpha_{12} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = [V_{12}]^{-1} \nu_R \tag{5.7}$$

it is possible to retrieve the values of α_1 and α_2. If both are positive numbers, then the relative velocity lies between ν_1 and ν_2 and a collision might happen (within a given prediction horizon). It is then necessary to introduce constraints to limit the vehicle motion.

5.1.2 Definition of the obstacle-free space

Following the work done in [12], the linear constraint introduced to limit the vehicle motion is defined as the line tangent to the intersection point of the relative velocity, with origin in $\hat{A}$, and the virtual box containing the pedestrian $\hat{B}$, as shown in figure 5.3.
To determine the intersection point, the relative velocity line is considered in

the implicit form, rather then $y = mx + q$, to consider the case of a velocity line parallel to the y axis.
Then, by considering the obstacle virtual box in the form

$$(x - x_c)^2 + (y - y_c)^2 = r_c^2 \tag{5.8}$$

with (x_c, y_c) and r_c being the center and the radius of the circular virtual box, respectively, it is possible to retrieve an analytical expression for the intersection point, that will be called (x_{sol}, y_{sol}). If two intersection points exist, then (x_{sol}, y_{sol}) is selected as the one closest to the vehicle.
The tangent is then retrieved as the line passing through the intersection point, with slope perpendicular to the line connecting the intersection point and the center of the circle $\hat{B}$; the linear constraint will be then expressed as:

$$h_x x + h_y y \leq l \tag{5.9}$$

where h_x and h_y are the coefficients related to the variables x and y, identifying the slope of the 1-dimensional line, while l is the constant term defining the distance from the origin, and are computed as:

$$\begin{cases} h_x = P_{1_x} - P_{ped_x} \\ h_y = P_{1_y} - P_{ped_y} \\ l = h_x P_{1_x} + h_y P_{1_y} \end{cases} \tag{5.10}$$

where P_1 is the intersection point on the pedestrian Virtual Box, while P_{ped} is the current position of the pedestrian. However, the definition of the personal space function allows to define a tighter constraint, by further reducing the free space for the wheelchair navigation:

$$h_x x + h_y y \leq l - \Delta l \tag{5.11}$$

In the obtained equation, Δl coefficient is computed based on the dimension of the personal space. This allows to correctly include the interaction between the pedestrian and the robot, aiming at satisfying requirements on human comfort. More specifically, $\Delta l = \Delta l_{sl} \sqrt{(h_x)^2 + (h_y)^2}$, and the coefficient Δl_{sl} represents the safety distance to impose so to properly account for the pedestrian personal space, and is evaluated according to Algorithm 1.

Algorithm 1 $safetyDistance$

$l_2 \leftarrow lineParameters(P_1, P_{ped})$;
$(P_2^1, P_2^2) \leftarrow findIntersection(l_2, reducedVB)$;
$P_2 \leftarrow findClosestPoint((P_2^1, P_2^2), P_r)$
$(P_{circle}^1, P_{circle}^2, P_{ellipse}^1, P_{ellipse}^2) \leftarrow findIntersection(l_2, PSf)$;
$(P_3^1, P_3^2) \leftarrow getIntersectionsPsf(P_{circle}^1, P_{circle}^2, P_{ellipse}^1, P_{ellipse}^2, P_{ped}, v_{ped})$
$P3 \leftarrow findClosestPoint((P_3^1, P_3^2), P_r)$
$\Delta l_{sl} \leftarrow euclidianDistance(P_2, P_3)$;
return Δl_{sl}

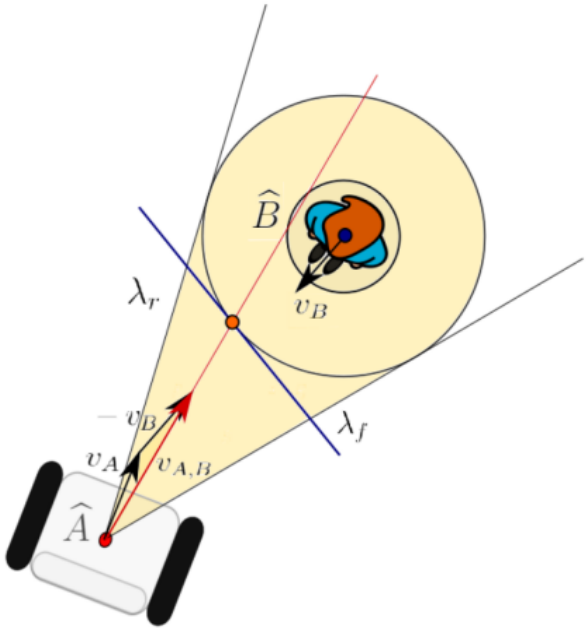


Figure 5.3: Position constraint computation (the blue line tangent to $\hat{B}$)

In the algorithm, P_1 is the intersection point on the pedestrian virtual box, $reducedVB$ is the pedestrian reduced Virtual Box and PSf is its Personal Space function.
Moreover, the following functions are used for the computation of the safety distance Δl_{sl}:

- $lineParameters$: gets as input the intersection between the relative velocity and the pedestrian Virtual Box, and the position of the pedestrian P_{ped}, providing as output the coefficients of the line l_2 between the two points, expressed as $ax + by + c = 0$;
- $findIntersection$: this function gets as input the line coefficients in l_1 and the parameters describing the pedestrian reduced Virtual Box, or its PSf, providing as output the intersection points. Due to the nature of these two geometric objects, two intersection points are determined. As the personal space function is obtained as the composition of a circle and an ellipse, respectively behind and in front of the pedestrian, with respect to its direction of motion: to adapt the $findIntersection$ function to PSf case, both the intersections on the circle and the ellipse are computed;
- $getIntersectionPSf$: gets as input the four intersection points obtained in the computation of the intersections for the PSf and determines the

correct intersection points. The function uses the pedestrian position P_{ped} and velocity v_{ped} as well;

- $findClosestPoint$: given as input two intersection points and the robot position P_r, it finds the one closer to the vehicle.

The resulting safety distance Δl_{sl} can be seen in Figure 5.4, as the distance between the two green points:

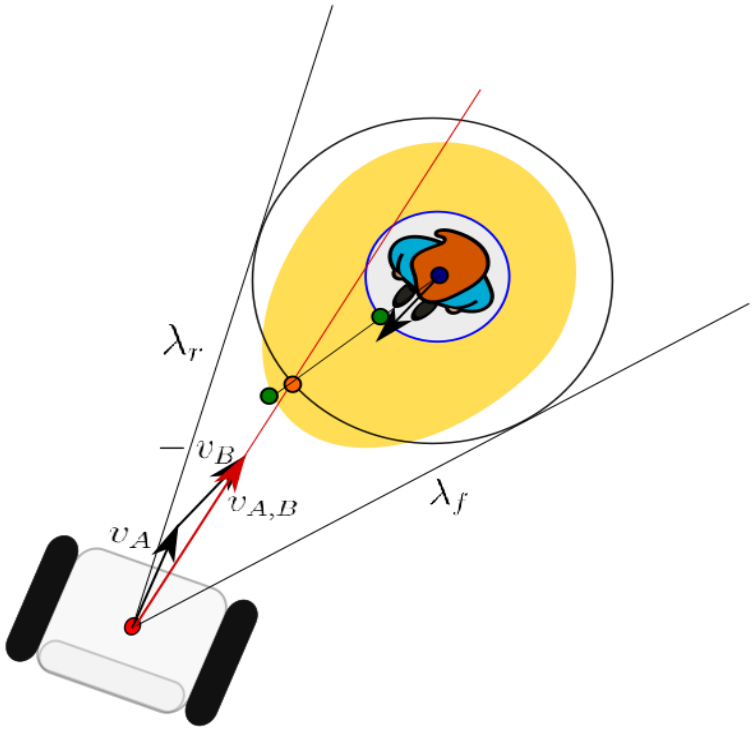


Figure 5.4: Δl_{sl} is determined as the euclidean distance between the points highlighted by the two green circles, representing the intersections between the pedestrian sensor virtual box (blue circle) and the Pedestrian personal space function (in yellow), respectively, with the black line between them defined obtained the pedestrian centre and the intersection between the relative velocity $v_{A,B}$ and the pedestrian virtual box enlarged to take into account the vehicles dimensions.

5.2 Model Predictive Control

Model Predictive Control is an advanced control method that exploits an explicit linear dynamic model of the system in order to predict the effect of the

control variables on the outputs. The main advantage of this approach, as highlighted in [6] is the possibility to express the control problem as an optimization problem over a finite horizon, called *Prediction Horizon*, subject to a specific set of constraints than can be imposed on input, state and output variables, to impose a desired behaviour on the system under control.
This specific formulation of the control problem makes the MPC approach particularly suitable for a large variety of applications.
Furthermore, the *Receiding Horizon* principle allows to obtain a more robust behaviour of the system, controlled as a closed-loop scheme.
In this work, a *convex optimization* problem is considered, with linear equality/inequality constraints and a quadratic convex cost function. In the following sections, the MPC problem related to the application of this thesis is introduced.

5.2.1 Vehicle model for control

As discussed in previous sections, to obtain a model suitable for a linear MPC approach, the Feedback Linearization has been introduced, which allows to rewrite the vehicle model as a linear decoupled system in the form:

$$\begin{cases} \dot{x}_P(t) = v_{P_x}(t) \\ \dot{y}_P(t) = v_{P_y}(t) \end{cases} \tag{5.12}$$

and, to make the model compatible with the discrete nature of the control approach, a discretization step through the forward Euler method is performed. This allows to obtain a discrete model of the system:

$$\begin{cases} x_P(k+1) = x_P(k) + \tau v_{P_x}(k) \\ y_P(k+1) = y_P(k) + \tau v_{P_y}(k) \end{cases} \tag{5.13}$$

or, in compact form:

$$\xi(k+1) = A\xi(k) + Bu(k) \tag{5.14}$$

where:

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \; B = \begin{bmatrix} \tau & 0 \\ 0 & \tau \end{bmatrix} \; \xi(k) = \begin{bmatrix} x_P(k) \\ y_P(k) \end{bmatrix} \; u(k) = \begin{bmatrix} v_{P_x}(k) \\ v_{P_y}(k) \end{bmatrix} \tag{5.15}$$

5.2.2 Cost function and Constraints

As previously described, the Model Predictive Control approach is based on the definition of a cost function: in this section, a finite horizon cost function specific for the wheelchair model is determined, to guarantee the tracking of a given reference trajectory, which can be expressed as:

$$\xi_{ref}(k+i) = \begin{bmatrix} x_{ref}(k+i) \\ y_{ref}(k+i) \end{bmatrix}$$

and fulfill the system requirements.
In this context, the cost function can be then expressed as follows:

$$J(\xi(k), u(t)) = \sum_{i=0}^{N-1} (\|\xi(k+i+1) - \xi_{ref}(k+i+1)\|^2_{Q,S} + \|u(k+i)\|^2_R) \quad (5.16)$$

where $\xi(k+i)$ is the i-th step ahead predicted state of the vehicle. Q, R and S are the weight matrices on state, control variable and terminal state, that must be properly tuned to ensure the fulfillment of requirements on state and control variables, as well as the system's stability. The expression

$$\|\xi(k+i+1) - \xi_{ref}(k+i+1)\|^2_{Q,S} \quad (5.17)$$

has been used to summarize the following sum:

$$\sum_{i=1}^{N-1} (\|\xi(k+i) - \xi_{ref}(k+i)\|^2_Q) + \|\xi(k+N) - \xi_{ref}(k+N)\|^2_S \quad (5.18)$$

To solve the optimization problem, a convenient way to formalize the control scheme is known as Open-Loop Solution and can be derived recalling the Lagrange equation:

$$\xi(k+i) = A^i \xi(k) + \sum_{j=0}^{i-1} A^{i-j-1} B u(k+j) \ , i > 0 \quad (5.19)$$

Setting the prediction horizon to N and letting:

$$\mathcal{A} = \begin{bmatrix} A \\ A^2 \\ A^3 \\ \vdots \\ A^N \end{bmatrix}_{(2N,2)} \qquad \mathcal{B} = \begin{bmatrix} B & 0 & \dots & 0 \\ AB & B & \dots & 0 \\ A^2B & AB & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{N-1}B & A^{N-2}B & \dots & B \end{bmatrix}_{(2N,2N)} \quad (5.20)$$

$$\Xi(k) = \begin{bmatrix} \xi(k+1) \\ \vdots \\ \xi(K+N) \end{bmatrix}_{(2N,1)} \qquad \mathcal{U}(k) = \begin{bmatrix} u(k) \\ \vdots \\ u(k+N-1) \end{bmatrix}_{(2N,1)} \quad (5.21)$$

it follows that:

$$\Xi(k) = \mathcal{A}\xi(k) + \mathcal{B}\mathcal{U}(k) \quad (5.22)$$

The cost function $J(\xi(k), u(k), k)$ can be then equivalently rewritten as:

$$\begin{aligned} J(\xi(k), \mathcal{U}(k)) = \|\Xi(k) - \Xi_{ref}(k)\|^2_{\mathcal{Q}} + \|\mathcal{U}(k)\|^2_{\mathcal{R}} = \\ (\Xi(k) - \Xi_{ref}(k))^T \mathcal{Q} (\Xi(k) - \Xi_{ref}(k)) + \mathcal{U}^T(k) \mathcal{R} \mathcal{U}(k) \end{aligned} \quad (5.23)$$

where matrices $\mathcal{Q}$ and $\mathcal{R}$ are defined as:

$$\mathcal{Q} = \begin{bmatrix} Q & & & \\ & \ddots & & \\ & & Q & \\ & & & S \end{bmatrix}_{(2N,2N)} \qquad \mathcal{R} = \begin{bmatrix} R & & \\ & \ddots & \\ & & R \end{bmatrix}_{(2N,2N)} \tag{5.24}$$

Recalling equation 5.22, the cost function becomes

$$J(k) = (\mathcal{A}\xi(k) + \mathcal{B}\mathcal{U}(k) - \Xi_{ref}(k))^T \mathcal{Q}(\mathcal{A}\xi(k) + \mathcal{B}\mathcal{U}(k) - \Xi_{ref}(k)) + \mathcal{U}^T(k)\mathcal{R}\mathcal{U}(k) \tag{5.25}$$

and by rearranging the obtained equation, it is ultimately possible to express the cost function as:

$$J = \mathcal{U}^T(k)H\mathcal{U}(k) + 2f^T\mathcal{U}(k) + cost \tag{5.26}$$

where $H = \mathcal{B}^T\mathcal{Q}\mathcal{B} + \mathcal{R}$ is called Hessian matrix and $f = (\mathcal{A}\xi(k) - \Xi_{ref})^T\mathcal{Q}\mathcal{B}$ is the gradient vector, while *cost* represents all the terms not depending from $\mathcal{U}(k)$. Another advantage in the formulation of the control problem as an optimization one is based on the fact that it is possible to directly include the constraints related to the state and the control variables, so that it is possible to take into account, in the research of a feasible solution, the boundaries that characterize a real context, i.e. the limitations on the actuators, the presence of forbidden areas or zones occupied by obstacles and, in this particular case, comfort or safety requirements. Such constraints can be expressed as:

$$\begin{aligned} \xi(k) &\in \mathbb{X} \subset \mathbb{R}^n \\ u(k) &\in \mathbb{U} \subset \mathbb{R}^m \end{aligned} \tag{5.27}$$

where:

$$\xi(k) = \begin{bmatrix} x_p(k) \\ y_p(k) \end{bmatrix} \quad u(k) = \begin{bmatrix} v_{px}(k) \\ v_{py}(k) \end{bmatrix} \tag{5.28}$$

Then, these constraints must be formulated in a form that is compatible with the optimization algorithm (i.e. convex equality or inequality constraints). In this work, convex programming is considered and linear constraints can be expressed as:

$$\begin{cases} A_{ineq_\xi}\Xi(k) \leq b_{ineq_\xi} \\ A_{eq_\xi}\Xi(k) = b_{eq_\xi} \\ \Xi_{min} \leq \Xi(k) \leq \Xi_{max} \end{cases} \qquad \begin{cases} A_{ineq_u}\mathcal{U}(k) \leq b_{ineq_u} \\ A_{eq_u}\mathcal{U}(k) = b_{eq_u} \\ \mathcal{U}_{min} \leq \mathcal{U}(k) \leq \mathcal{U}_{max} \end{cases} \tag{5.29}$$

where matrices A_{ineq} and A_{eq} express the number of constraints to be imposed, b_{ineq} and b_{eq} are vector of constant terms whose length equals the number of row of the two described matrices, while Ξ_{min}, Ξ_{max}, $\mathcal{U}_{min}$ and $\mathcal{U}_{max}$ are the lower and upper bounds for state and control variables respectively. In the following sections, limitations of the actuators and comfort requirements are introduced, enforcing constraints on the maximum longitudinal velocity of the vehicle, as well as constraints on acceleration and vehicle position during navigation, with reference to the work done in [12].

5.2.3 Velocity constraints

To take into account actuator limitations and comfort requirements, maximum velocity constraints are enforced on the vehicle longitudinal speed v, together with constraints on velocity variation.
In this work, a set of linear constraints on the vehicle longitudinal velocity is obtained, exploiting the relationships obtained through the feedback linearization procedure:

$$\begin{cases} v(t) = v_{P_x}(t)\cos\theta(t) + v_{P_y}(t)\sin\theta(t) \\ \omega(t) = \frac{1}{\varepsilon}(v_{P_y}(t)\cos\theta(t) - v_{P_x}(t)\sin\theta(t)) \end{cases} \tag{5.30}$$

where ε is the distance from the wheels axle of the point P used in the feedback linearization of the vehicle's kinematic model. Recalling the Differential Drive kinematic relationships:

$$v(t) = R\frac{\omega_R(t) + \omega_L(t)}{2} \qquad \omega(t) = R\frac{\omega_R(t) - \omega_L(t)}{d} \tag{5.31}$$

with R and d being the wheels' radius and wheels' separation respectively, the following relationships for wheels rotational speed can be obtained:

$$\omega_R(t) = \frac{2v(t) + d\omega(t)}{2R} \qquad \omega_L(t) = \frac{2v(t) - d\omega(t)}{2R} \tag{5.32}$$

and by combining these with Eq. 5.30:

$$\begin{aligned} \omega_R(t) &= \frac{1}{2R}(2\cos\theta(t) - \frac{d}{\varepsilon}\sin\theta(t))v_{P_x}(t) + \frac{1}{2R}(2\sin\theta(t) + \frac{d}{\varepsilon}\cos\theta(t))v_{P_y}(t) \\ \omega_L(t) &= \frac{1}{2R}(2\cos\theta(t) + \frac{d}{\varepsilon}\sin\theta(t))v_{P_x}(t) + \frac{1}{2R}(2\sin\theta(t) - \frac{d}{\varepsilon}\cos\theta(t))v_{P_y}(t) \end{aligned} \tag{5.33}$$

It is then possible to impose constraints on the control variables of the linearized system ,v_{P_x} and v_{P_y}, by limiting the rotational velocities of the two wheels:

$$\bar{\omega}_m \leq \omega_R \leq \bar{\omega}_M \qquad \bar{\omega}_m \leq \omega_L \leq \bar{\omega}_M \tag{5.34}$$

where $\bar{\omega}_m$ and $\bar{\omega}_M$ are the minimum and maximum values of the wheels rotational velocity. By selecting then as proper maximum longitudinal velocity v_{max}, it is possible to define $\bar{\omega}_m$, $\bar{\omega}_M$ as:

$$\bar{\omega}_M = \frac{v_{max}}{R} \qquad \bar{\omega}_m = \frac{v_{min}}{R} \tag{5.35}$$

with $v_{min} = -v_{max}$.
Then, assuming θ (the state variable representing the wheelchair orientation) to be known, the previous constraints defined by Eq. 5.33 can be rewritten as:

$$\frac{1}{2R}\begin{bmatrix} 2\cos\theta(t) - \frac{d}{\varepsilon}\sin\theta(t) & 2\sin\theta(t) + \frac{d}{\varepsilon}\cos\theta(t) \\ 2\cos\theta(t) + \frac{d}{\varepsilon}\sin\theta(t) & 2\sin\theta(t) - \frac{d}{\varepsilon}\cos\theta(t) \end{bmatrix}\begin{bmatrix} v_{P_x} \\ v_{P_y} \end{bmatrix} \leq \begin{bmatrix} \bar{\omega}_M \\ \bar{\omega}_M \end{bmatrix} \tag{5.36}$$

$$-\frac{1}{2R}\begin{bmatrix}2\cos\theta(t)-\frac{d}{\varepsilon}\sin\theta(t) & 2\sin\theta(t)+\frac{d}{\varepsilon}\cos\theta(t)\\ 2\cos\theta(t)+\frac{d}{\varepsilon}\sin\theta(t) & 2\sin\theta(t)-\frac{d}{\varepsilon}\cos\theta(t)\end{bmatrix}\begin{bmatrix}v_{P_x}\\ v_{P_y}\end{bmatrix}\leq-\begin{bmatrix}\bar{\omega}_m\\ \bar{\omega}_m\end{bmatrix} \tag{5.37}$$

Then, defining:

$$\bar{A}_i=\frac{1}{2R}\begin{bmatrix}2\cos\theta(t)-\frac{d}{\varepsilon}\sin\theta(t) & 2\sin\theta(t)+\frac{d}{\varepsilon}\cos\theta(t)\\ 2\cos\theta(t)+\frac{d}{\varepsilon}\sin\theta(t) & 2\sin\theta(t)-\frac{d}{\varepsilon}\cos\theta(t)\end{bmatrix}\quad \bar{b}=\begin{bmatrix}\bar{\omega}_M\\ \bar{\omega}_M\\ -\bar{\omega}_m\\ -\bar{\omega}_m\end{bmatrix} \tag{5.38}$$

it is possible to rewrite the previous equations in matrix form:

$$\begin{bmatrix}\bar{A}_i\\ -\bar{A}_i\end{bmatrix}_{(4,2)}\begin{bmatrix}v_{P_x}\\ v_{P_y}\end{bmatrix}_{(2,1)}\leq\bar{b}_{(4,1)} \tag{5.39}$$

To be compatible with the optimization problem formulation, such constraints must be defined along the entire prediction horizon ($\forall i\in\{0,\ldots,N-1\}$):

$$\bar{A}_{vel}\mathcal{U}(k)\leq\bar{b}_{vel} \tag{5.40}$$

with:

$$\bar{A}_{vel}=\begin{bmatrix}\bar{A}_1 & 0 & \ldots & 0\\ 0 & \bar{A}_2 & \ldots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \ldots & \bar{A}_N\\ -\bar{A}_1 & 0 & \ldots & 0\\ 0 & -\bar{A}_2 & \ldots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \ldots & -\bar{A}_N\end{bmatrix}_{(4N,2N)}\qquad \bar{b}_{vel}=\begin{bmatrix}\bar{\omega}_M\\ \vdots\\ \vdots\\ \bar{\omega}_M\\ -\bar{\omega}_m\\ \vdots\\ \vdots\\ -\bar{\omega}_m\end{bmatrix}_{(4N,1)} \tag{5.41}$$

The values of $\theta(i)$ can be reasonably estimated considering the values of the control sequence determined at the previous time instant, $\mathcal{U}(k-1)$. More specifically, it is possible to determine:

$$\hat{\theta}(k+i|k-1)=\begin{bmatrix}\theta(k)\\ \theta(k+1|k-1)\\ \vdots\\ \theta(k+N-1|k-1)\end{bmatrix}$$

where $\hat{\theta}(k+i|k-1)$ is the predicted vehicle heading obtained from the optimal control sequence determined at time instant $k-1$ except for the first element, corresponding to the actual measurement of the vehicle heading.

5.2.4 Velocity variation constraints

To take into account the discrete-time nature of the Model Predictive Control approach, the velocity variation at any time instant k can be expressed as:

$$\Delta v(k) = v(k) - v(k-1) \tag{5.42}$$

Moreover, the constraint can be imposed by limiting the velocity variation between two consecutive time instants:

$$-\Delta v_{max} \leq \Delta v(k) \leq \Delta v_{max} \tag{5.43}$$

The boundary value Δv_{max} can be evaluated considering vehicle limitations. More specifically, considering the avarage of acceleration between two consecutive iterations:

$$\bar{a} = \frac{\Delta v}{\Delta t} \tag{5.44}$$

the maximum velocity variation can be derived by taking into account the product between the controller sampling time τ and the maximum value of the average acceleration $\bar{a}_{max}$:

$$\Delta v_{max} = \bar{a}_{max}\tau \tag{5.45}$$

Due to the decoupling of the control variables along the x and y axes obtained from the feedback linearization procedure, constraints on the velocity variation at time instant $k+i$ are decoupled and can be imposed directly on the control variables v_{P_x}, v_{P_y} of the linearized system:

$$\begin{aligned} -\Delta v_{max} \leq v_{P_x}(k+i) - v_{P_x}(k+i-1) \leq \Delta v_{max} \\ -\Delta v_{max} \leq v_{P_y}(k+i) - v_{P_y}(k+i-1) \leq \Delta v_{max} \end{aligned} \tag{5.46}$$

and considering the velocity variation constraints along the entire prediction horizon ($\forall i \in \{0, \ldots, N-1\}$), the following equation in matrix form is obtained:

$$A_{var}(V\mathcal{U}(k) - v_0) \leq \Delta V \tag{5.47}$$

with:

$$A_{var} = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \\ -1 & & \\ & \ddots & \\ & & -1 \end{bmatrix}_{(4N,2N)} \qquad \Delta V = \begin{bmatrix} \Delta v_{max} \\ \vdots \\ \Delta v_{max} \end{bmatrix}_{(4N,1)}$$

$$v_0 = \begin{bmatrix} v_{P_x}(k-1) \\ v_{P_y}(k-1) \\ 0 \\ \vdots \\ 0 \end{bmatrix}_{(2N,1)} \qquad V = \begin{bmatrix} I_{(2)} & & & \\ -I_{(2)} & \ddots & & \\ & \ddots & \ddots & \\ & & -I_{(2)} & I_{(2)} \end{bmatrix}_{(2N,2N)} \tag{5.48}$$

where $I_{(2)}$ is the identity matrix of order 2. Ultimately, recalling that in quadratic programming linear inequality constraints, along the entire prediction horizon, can be expressed in the form:

$$A_{ineq}\mathcal{U}(k) \leq b_{ineq} \tag{5.49}$$

the condition to be imposed on velocity variation must be written as:

$$A_{\Delta V}\mathcal{U}(k) \leq b_{\Delta V} \tag{5.50}$$

By rearranging Eq. 5.47, matrices $A_{\Delta V}$ and $b_{\Delta V}$ can be obtained as:

$$A_{\Delta V} = [A_{var}V]_{(4N,2N)} \qquad b_{\Delta V} = [\Delta V + A_{var}v_0]_{(4N,1)} \tag{5.51}$$

5.2.5 Position constraints

As the state of the system coincides with the position of the vehicle, enforcing a constraint on the state variables in the form:

$$x(k) \in \mathbb{X} \tag{5.52}$$

defines a trust region representing the space of the environment in which the robot is allowed to move.
As already described, the concept of Velocity Obstacles is applied in this context considering the obstacle virtual box, so to define a set of hyperplanes that separate the convex region representing the free space (and containing the wheelchair) from the detected obstacles, and hence avoiding collisions.
Considering vehicle configuration space, the hyperplanes that delimit the trust region has the form:

$$h_x x + h_y y \leq l \tag{5.53}$$

where h_x and h_y are the coefficients related to the variables x and y, identifying the slope of the 1-dimensional line, while l is the constant term defining the distance from the origin. Each equation is implemented in the MPC as a linear constraint dividing the state space into two half planes, one representing the admissible space, while the other the forbidden space. These linear constraints are computed for each time instant within the prediction horizon, so to properly define an obstacle-free region of space for the future predictions of the vehicle motion.

Position constraints implementation

Constraints related to the vehicle position, expressed as in Eq. 5.53, can then be defined, for each time instant $k+i$ within the selected prediction horizon N, as a set of linear constraints of the kind:

$$h_x^{(i)}x(k+i) + h_y^{(i)}y(k+i) \leq l^{(i)} \qquad i = 1, \ldots, N \tag{5.54}$$

where $h_x^{(i)}$ and $h_y^{(i)}$ represent the coefficients for the constraint at instant $k+i$ for a given pedestrian, meaning that, inside a prediction horizon, different constraints can be defined, according to the position and relative speed at each time instant.
Moreover, it is important to ensure that the half plane denoting the free space is the one in which the estimated position of the vehicle is contained, or equivalently that the pedestrian is contained in the half plane denoting the forbidden space. This condition is checked at any time instant within the prediction horizon, and to do so, the following conditions must hold:

$$h_x^{(i)} x_{ped}(k+i) + h_y^{(i)} y_{ped}(k+i) \geq l^{(i)} \qquad i = 1, \ldots, N \tag{5.55}$$

where $x_{ped}(k+i)$ $y_{ped}(k+i)$ are the coordinates of a given pedestrian at instant $k+i$, inside the prediction horizon, predicted according to a proper algorithm, starting from the available measurements. If this is not the case, in order to obtain an inequality with the lower or equal sign, suitable for the MPC implementation, it is necessary to invert the signs (i.e., selecting the other half plane delimited by the line), obtaining:

$$(-h_x^{(i)}) x(k+i) + (-h_y^{(i)}) y(k+i) \leq (-l^{(i)}) \qquad i = 1, \ldots, N \tag{5.56}$$

As for the control variables case, in quadratic programming, linear system state constraints along the entire prediction horizon must be expressed as linear inequalities with respect to the vector of control variables.
Position constraints along the entire prediction horizon can then be rewritten in a more suitable formulation for the MPC:

$$\underbrace{\begin{bmatrix} h_{k+1|k-1} & & \\ & \ddots & \\ & & h_{k+N|k-1} \end{bmatrix}}_{H_{obs}} \underbrace{\begin{bmatrix} \xi(k+1) \\ \vdots \\ \xi(k+N) \end{bmatrix}}_{\Xi(k)} \leq \underbrace{\begin{bmatrix} l_{k+1|k-1} \\ \vdots \\ h_{k+N|k-1} \end{bmatrix}}_{L_{obs}} \tag{5.57}$$

In this case, the subscripts $k+1|k-1, \ldots, k+N|k-1$ express the fact that the constraint will be computed for the whole prediction horizon, by taking into account the state prediction obtained from the optimal control sequence computed at time instant $k-1$.
However, at every time instant the robot can access only its current position, but for the computation of position constraints along the prediction horizon, an estimate of future robot positions is required. This problem can be solved

recalling the Lagrange equation:

$$\Xi(k) = \begin{bmatrix} A \\ A^2 \\ A^3 \\ \vdots \\ A^N \end{bmatrix}_{(2N,2)} \xi(k-1) + \begin{bmatrix} B & 0 & \dots & 0 \\ AB & B & \dots & 0 \\ A^2B & AB & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{N-1}B & A^{N-2}B & \dots & B \end{bmatrix}_{(2N,2N)} \mathcal{U}(k-1) \tag{5.58}$$

in which vector $\Xi(k)$ represents the predicted future positions of the vehicle, based on the optimal control sequence $\mathcal{U}(k-1)$ and the previous position $\xi(k-1)$. $\mathcal{U}(k-1)$ is also the vector of linear velocities that will be used for the computation of constraints, through the process described in the previous section, discarding the first pair of elements, representing the linear velocities components that led the robot to its current position. The resulting vector of predicted linear velocities is then:

$$\mathcal{U}_{pred}(k) = \begin{bmatrix} u(k) \\ u(k+1) \\ \vdots \\ u(k+N-2) \end{bmatrix}_{(2(N-1),1)} \tag{5.59}$$

with:

$$u(k+i) = \begin{bmatrix} v_{p_x}(k+i) \\ v_{p_y}(k+i) \end{bmatrix} \qquad i \geq 0 \tag{5.60}$$

meaning that only $N-1$ pairs of linear velocities are available. Since only the last pair of components is missing, it is reasonable to assume that the robot will keep the same linear velocities in the last step of the prediction horizon, so that the vector of predicted linear velocities can be fully defined as:

$$\mathcal{U}_{pred}(k) = \begin{bmatrix} u(k) \\ u(k+1) \\ \vdots \\ u(k+N-2) \\ u(k+N-2) \end{bmatrix}_{(2N,1)} \tag{5.61}$$

The linear velocities $\mathcal{U}_{pred}(k)$ and the predicted future vehicle linear positions $\Xi(k)$ are then used to estimate the geometric center of the vehicle, that will be used to compute the constraints coefficients. In conclusion, position constraints can be expressed in compact form as:

$$[H_{obs}]_{(N,2N)}[\Xi(k)]_{(2N,1)} \leq [L_{obs}]_{(N,1)} \tag{5.62}$$

and recalling Eq. 5.22:

$$H_{obs}\underbrace{(\mathcal{A}\xi(k)+\mathcal{B}\mathcal{U}(k))}_{\Xi(k)} \leq L_{obs} \tag{5.63}$$

from which:

$$H_{obs}\mathcal{B}\mathcal{U}(k) \leq L_{obs} - H_{obs}\mathcal{A}\xi(k) \tag{5.64}$$

Ultimately, by defining:

$$A_{obs} = [H_{obs}\mathcal{B}]_{(N,2N)} \qquad b_{obs} = [L_{obs} - H_{obs}\mathcal{A}\xi(k)]_{(N,1)} \tag{5.65}$$

it is possible to rewrite the constraint equations as:

$$A_{obs}\mathcal{U}(k) \leq b_{obs} \tag{5.66}$$

To include the presence of more then one obstacle in the constraint formulation, it is possible to define $A_{obs}^{(j)}$ and $b_{obs}^{(j)}$ as the constraint related to the j-th obstacle, so that for a number $n_{obs} > 0$ of obstacles it is possible to write:

$$\begin{bmatrix} A_{obs}^{(1)} \\ \vdots \\ A_{obs}^{(n_{obs})} \end{bmatrix}_{(n_{obs}N,2N)} \qquad \begin{bmatrix} b_{obs}^{(1)} \\ \vdots \\ b_{obs}^{(n_{obs})} \end{bmatrix}_{(n_{obs}N,1)} \tag{5.67}$$

A relevant aspect to consider in the computation of position constraints is that even though such conditions determine an obstacle-free subspace of the whole state space in which the robot can move, it might happen that the current predicted vehicle positions, used to compute the inequalities defining the constraints inside the prediction horizon, cause a collision with one of the obstacle in the environment. As highlighted in figure 5.5 representing a case with a fixed obstacle at time instant k, positions $\Xi(k)$ might fall inside the forbidden space for the vehicle. Recalling the method for the computation of position constraints already described, if the vehicle position is inside the virtual box the tangent line cannot be determined, while in such condition a constraint should be defined. To avoid this issue,the constraint coefficients h_x, h_y, l computed at time instant $k+i$, with $i = 1, \ldots, N-1$ are saved and, if the robot position at the next time instant $k+i+1$ is inside the pedestrian virtual box, then the last available computed constraint is applied. Otherwise, new constraint coefficients are computed and saved. By defining:

- $currPose$: variable containing the vehicle current pose (cartesian position and heading);
- U_{pred}: optimal control sequence computed at time instant $k-1$;

- $(x_{pred}, y_{pred}, v_{long})$: predicted values of the vehicle position and longitudinal velocity inside the prediction horizon;
- v_{rel}: relative velocity between the vehicle and the considered obstacle;
- $robotInsideObstacle$: variable used to check whether the vehicle predicted position falls inside the obstacle Virtual Box;
- α_{12}, V_{12}: variables defined in section 5.1.1, used in the collision detection procedure;
- VB: pedestrian Virtual Box parameters;

The constraint coefficients computation procedure, for one obstacle, is summarized in Algorithm 2:

Algorithm 2 *Computation of position constraints*

$(x_{pred}, y_{pred}, v_{long}) \leftarrow prediction(\Xi(k), currPose, U_{pred})$;
$(h_{x_{temp}}, h_{y_{temp}}, l_{temp}) \leftarrow (0, 0, 0)$
for $i = 1, i \leq N, i++$ **do**
  $v_{rel} \leftarrow v_{long}(i) - v_{ped}(i)$;
  $(V_{12}, robotInsideObst) \leftarrow findTangents(x_{pred}(i), y_{pred}(i), VB)$;
  if $robotInsideObst = False$ **then**
    $\alpha_{12} \leftarrow V_{12}/v_{rel}$
    if $\alpha_{12}(1) > 0$ *and* $\alpha_{12}(1) > 0$ **then**
      $(h_x(i), h_y(i), l(i)) \leftarrow constaintsComp(VB, v_{rel})$;
      $(h_{x_{temp}}, h_{y_{temp}}, l_{temp}) \leftarrow (h_x(i), h_y(i), l(i))$;
    end if
  else
    $(h_x(i), h_y(i), l(i)) \leftarrow (h_{x_{temp}}, h_{y_{temp}}, l_{temp})$;
  end if
end for

The function *constraintsComp*, described in Algorithm 3 uses the following subfunctions to obtain the coefficients for the linear constraints:

- $lineParameters$: gets as input the relative velocity between the vehicle and the pedestrian and the vehicle position P_r, providing as output the coefficients of the line obtained as the extension of the relative velocity vector, in the form $ax + by + c = 0$;
- $findIntersection$: given as input the line parameters previously determined and the pedestrian VB (or the Personal Space Function) finds the intersection point between them; due to the nature of the pedestrian VB (and Personal Space function as well) more than one point is found;
- $findClosestPoint$: given the two intersection points, determines the one closer to the robot;

- *constraintCoefficients*: given the intersection point P_1, and the current pedestrian position P_{ped}, finds the coefficients h_x, h_y, l of the linear inequality, as described in section 5.1.2.

Algorithm 3 *constraintComp*

$l_1 \leftarrow lineParameters(v_rel, P_r)$;
$(P_1^1, P_1^2) \leftarrow findIntersection((a, b, c), VB)$;
$P_1 \leftarrow findClosestPoint((P_1^1, P_1^2), P_r)$
$(h_x(i), h_y(i), l(i)) \leftarrow constraintCoefficients(P1, P_{ped})$;
return $(h_x(i), h_y(i), l(i))$

The function $findTangents$ instead follows the procedure described in the Collision Detection section.

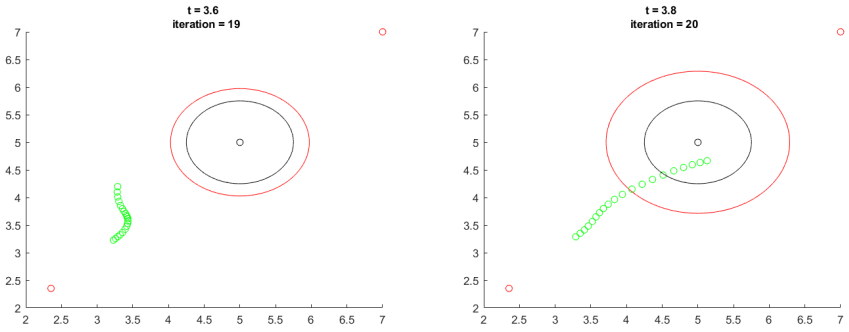


Figure 5.5: The figure shows two subsequent prediction steps of the simulation, where the green circles represent the motion of the robot geometric center, the black circle is the obstacle virtual box and the bigger red circle is its Personal Space function. Due to motion of the wheelchair in iteration 19, some of the predicted robot positions used in iteration 20 are unconstrained, falling in the obstacle

5.3 Slack variables

It may happen that the constraints defined in the previous sections guarantee safety on one side, but also bounds that are too strict, leading possibly to the unfeasibility of the problem. This can be avoided exploiting the notion of *hard* and *soft* constraints, through the introduction of slack variables in the optimization problem, that allow for a constraint relaxation when required.

5.3.1 Cost function

Slack variables can be considered as part of the optimization variables. It is hence necessary to make appropriate modifications to the problem formulation, to meet the new requirements. To this aim, a new cost function must be defined:

$$\bar{J}(k) = J(k) + \|u_{sl_p}(k)\|^2_{S_p} + \|u_{sl_a}(k)\|^2_{S_a} \tag{5.68}$$

where S_p and S_a are additional weight matrices, while $u_{sl_p}(k)$ and $u_{sl_a}(k)$ are two sets of slack variables. In particular:

- $u_{sl_p}(k)$ is a vector of dimension n_{obs}, i.e. the number of obstacles, related to the slack on the position constraints;
- $u_{sl_a}(k)$ is a vector of two elements, related to the slack on the acceleration constraints along the of the global frame, Δv_{sl_x} and Δv_{sl_y}.

The weight matrices are then defined as:

$$S_p = \begin{bmatrix} s_p & & \\ & \ddots & \\ & & s_p \end{bmatrix}_{(n_{obs},n_{obs})} \qquad s_a = \begin{bmatrix} s_a & \\ & s_a \end{bmatrix}_{(2,2)} \tag{5.69}$$

and the values of parameters s_p and s_a will be chosen to allow the relaxation of constraints only when needed. More specifically, by selecting a high value of weights s_p and s_a with respect to q and r, the controller will produce solutions with values of the slack variables different from zero only if the feasibility of the problem is compromised.
Ultimately, by recalling the definition of the quadratic cost function, it is possible to define:

$$\bar{J}(k) = \frac{1}{2}\bar{\mathcal{U}}(k)^T \bar{H} \bar{\mathcal{U}}(k) + 2\bar{f}^T \bar{\mathcal{U}}(k) + cost \tag{5.70}$$

where:

$$\bar{H} = \begin{bmatrix} H & & \\ & S_p & \\ & & S_a \end{bmatrix}_{(2N+n_{obs}+2,2N+n_{obs}+2)} \tag{5.71}$$
$$\bar{f}^T = \begin{bmatrix} f^T & 0_{(1,n_{obs})} & 0_{(1,2)} \end{bmatrix}_{(1,2N+n_{obs}+2)}$$

and:

$$\bar{\mathcal{U}}(k) = \begin{bmatrix} \mathcal{U}(k) \\ u_{sl_p}(k) \\ u_{sl_a}(k) \end{bmatrix}_{(2N+n_{obs}+2,1)} \tag{5.72}$$

5.3.2 Soft velocity variation constraints

During the vehicle navigation it may happen that, due to the hard bound on acceleration, the feasibility of the optimization problem could be compromised. Moreover, the introduction of such variables allows to slightly exceed the velocity variation constraints previously defined‘, in conditions where it is required to avoid an obstacle. In spite of the importance of acceleration constraints, for safety and comfort requirements, it is necessary to define a constraint relaxation on both acceleration variables. To formalize this condition, it is possible to consider the following constraints:

$$\begin{aligned} v_x(k+i) - v_x(k+i-1) - \Delta v_{sl_x}(k) &\leq \Delta v_{max} \\ v_x(k+i-1) - v_x(k+i) + \Delta v_{sl_x}(k) &\leq \Delta v_{max} \\ v_y(k+i) - v_y(k+i-1) - \Delta v_{sl_y}(k) &\leq \Delta v_{max} \\ v_y(k+i-1) - v_y(k+i) + \Delta v_{sl_y}(k) &\leq \Delta v_{max} \\ \forall i \in \{0, \dots, N-1\} & \end{aligned} \tag{5.73}$$

where the slack variables $\Delta v_{sl_x}(k)$ and $\Delta v_{sl_y}(k)$ will impose an increment on the acceleration, if the problem does not admit any feasible solution.
To guarantee safety during navigation, it is important to impose the following constraints on the slack variables:

$$\begin{aligned} 0 \leq \Delta v_{sl_x}(k) \leq \Delta v_{sl_{max}} \\ 0 \leq \Delta v_{sl_y}(k) \leq \Delta v_{sl_{max}} \end{aligned} \tag{5.74}$$

that limits the increment of the acceleration to the threshold $\Delta v_{sl_{max}}$.
The value for parameter $\Delta v_{sl_{max}}$ can be determined following the same procedure described for the velocity variation constraints, hence:

$$\Delta v_{sl_{max}} = \bar{a}_{sl_{max}} \tau \tag{5.75}$$

where $\bar{a}_{sl_{max}}$ can be chosen with the same value as $\bar{a}_{max}$.
It is then possible to rewrite the constraint equations, in matrix form, as:

$$\bar{A}_{\Delta V} \mathcal{U}(k) \leq b_{\Delta V} \tag{5.76}$$

where:

$$\bar{A}_{\Delta v} = [A_{\Delta v} \; A_{\Delta_{sl}}]_{(4N, 2N+n_{obs}+2)} \tag{5.77}$$

in which matrix $A_{\Delta_{sl}}$ can be defined as:

$$A_{\Delta sl} = \begin{bmatrix} 0 & \dots & 0 & \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \\ 0 & \dots & 0 & \\ & \vdots & & \vdots \\ 0 & \dots & 0 & \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \\ 0 & \dots & 0 & \\ \underbrace{\qquad\qquad}_{u_{sl\_p}(k)} & & & \underbrace{\qquad\qquad}_{u_{sl\_a}(k)} \end{bmatrix}_{(4N, n_{obs}+2)}$$

that allows to nullify the contribution of slack variables on position constraints, that are not relevant in this context.

5.3.3 Soft position constraints

As already described in section 5.1.2, it is possible to rewrite position constraints as

$$h_x x + h_y y \leq l - \Delta l \tag{5.78}$$

where $\Delta l = \Delta l_{sl}\sqrt{(h_x)^2 + (h_y)^2}$. Considering the $safetyDistance$ function described in Algorithm 1, the constraints computation function then has to be modified as follows:

Algorithm 4 *constraintComp*

$l_1 \leftarrow lineParameters(v_{rel}, P_r)$;
$(P_1^1, P_1^2) \leftarrow findIntersection(l_1, VB)$;
$P_1 \leftarrow findClosestPoint((P_1^1, P_1^2), P_r)$
$(h_x(i), h_y(i), l(i)) \leftarrow constraintCoefficients(P_1, P_{ped})$;
$\Delta l_{sl} \leftarrow safetyDistance(P_1, P_{ped}, P_r, v_{ped}, PSf)$
$\Delta l(i) \leftarrow \Delta l_{sl}\sqrt{(h_x(i))^2 + (h_y(i))^2}$
return $(h_x(i), h_y(i), l(i), \Delta l(i))$

Through the definition of position constraints expressed in Eq. 5.78, and by considering a generic number of obstacles n_{obs}, it is possible to rewrite the constraints to account for the safety distance, which could eventually be violated by the optimizer if needed, as:

$$\begin{aligned} h_x^{(i)(j)} x(k+i) + h_y^{(i)(j)} y(k+i) \leq l^{(i)(j)} - \Delta l^{(i)(j)}(1 - u_{sl_p}^{(j)}(k)) \\ \forall i \in \{1, \dots, N\},\ \forall j \in \{1, \dots, n_{obs}\} \end{aligned} \tag{5.79}$$

where:

$$\Delta l^{(i)(j)} = \Delta l_{sl}^{(i)(j)} \sqrt{(h_x^{(i)(j)})^2 + (h_y^{(i)(j)})^2} \tag{5.80}$$

so that, by defining an extraction matrix $E^{(j)}$ as:

$$E^{(j)} = \begin{bmatrix} 0 & \dots & 0 & \dots & -\Delta l^{(i)(j)} & 0 & \dots & 0 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \\ 0 & \dots & 0 & \dots & -\Delta l^{(i)(j)} & 0 & \dots & 0 \end{bmatrix} \tag{5.81}$$

where the only column with values different from zero is the one in position j, it is possible to formulate:

$$\begin{aligned} & u_{sl_p}^{(j)}(k) = E^{(j)} u_{sl_p}(k) \\ & 0 \leq u_{sl_p}^{(j)}(k) \leq 1 \quad \forall j \in \{1, \dots, n_{obs}\} \end{aligned} \tag{5.82}$$

where $u_{sl_p}(k)$ is the vector of slack variables. The vehicle will then try to keep a distance from the j-th obstacle described by the relationship:

$$h_x^{(i)(j)} x(k+i) + h_y^{(i)(j)} y(k+i) \leq l^{(i)(j)} - \Delta l^{(i)(j)}, \quad if\ u_{sl_p}^{(j)} = 0 \tag{5.83}$$

while the limit condition for the admissible space is defined by:

$$h_x^{(i)(j)} x(k+i) + h_y^{(i)(j)} y(k+i) \leq l^{(i)(j)}, \quad if\ u_{sl_p}^{(j)} = 1 \tag{5.84}$$

Then, by considering the position constraints in matrix form:

$$[H_{obs}^{(j)}]_{(N,2N)} [\Xi(k)]_{(2N,1)} \leq [\bar{L}_{obs}^{(j)}]_{(N,1)} \tag{5.85}$$

where

$$\bar{L}_{obs}^{(j)} = L_{obs}^{(j)} - \begin{bmatrix} \Delta l^{(j)} \\ \vdots \\ \Delta l^{(j)} \end{bmatrix}_{(N,1)} \tag{5.86}$$

and recalling the definition of the enlarged vector of control variables:

$$\bar{\mathcal{U}}(k) = \begin{bmatrix} \mathcal{U}(k) \\ u_{sl_p}(k) \\ u_{sl_a}(k) \end{bmatrix}_{(2N+n_{obs}+2,1)} \tag{5.87}$$

it is possible to express the condition (2.68) with respect to the set of control variables:

$$\bar{A}_{obs}^{(j)} \bar{\mathcal{U}}(k) \leq \bar{b}_{obs}^{(j)} \tag{5.88}$$

with:

$$\bar{A}_{obs}^{(j)} = [H_{obs}^{(j)} \mathcal{B} \quad E^{(j)} \quad 0_{(N,2)}]_{(N,2N+n_{obs}+2)} \qquad \bar{b}_{obs}^{(j)} = \bar{L}_{obs}^{(j)} - H_{obs}^{(j)} \mathcal{A} \xi(k) \tag{5.89}$$

and, as discussed for soft velocity variation constraints, the element $0_{(N,2)}$ allows to nullify the contribution of acceleration slack variables. In conclusion, it is possible to define the full set of position constraints as:

$$\bar{A}_{obs} = \begin{bmatrix} \bar{A}_{obs}^{(1)} \\ \vdots \\ \bar{A}_{obs}^{(n_{obs})} \end{bmatrix}_{(n_{obs}N,2N+n_{obs}+2)} \qquad \bar{b}_{obs} = \begin{bmatrix} \bar{b}_{obs}^{(1)} \\ \vdots \\ \bar{b}_{obs}^{(n_{obs})} \end{bmatrix}_{(n_{obs}N,1)} \tag{5.90}$$

5.3.4 Slack variables constraints

As described in the previous sections, the introduction of soft position and acceleration constraints in the problem formulation requires a limitation on the values of the slack variables. In particular, constraint on position slack variable was defined as:

$$0 \leq u_{sl_p}^{(j)}(k) \leq 1 \quad \forall j \in \{1, \dots, n_{obs}\} \tag{5.91}$$

while for slack variables on the vehicle acceleration, the following constraints have been introduced:

$$\begin{aligned} 0 \leq \Delta v_{sl_x}(k) \leq \Delta v_{sl_{max}} \\ 0 \leq \Delta v_{sl_y}(k) \leq \Delta v_{sl_{max}} \end{aligned} \tag{5.92}$$

that force the increment on maximum acceleration to assume a value that guarantees safety, but satisfies the actuation limitations as well.
By extending the previous conditions along the entire prediction horizon:

$$A_{sl\_p} = \begin{bmatrix} 0 & \dots & 0 & 1 & \dots & 0 & 0 & 0 \\ \vdots & & \vdots & & \ddots & & \vdots & \vdots \\ 0 & & 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & & 0 & -1 & \dots & 0 & 0 & 0 \\ \vdots & & \vdots & & \ddots & & \vdots & \vdots \\ 0 & \dots & 0 & 0 & \dots & -1 & 0 & 0 \\ \underbrace{\qquad\qquad}_{\mathcal{U}(k)} & & & \underbrace{\qquad\qquad}_{u_{sl\_p}(k)} & & & \underbrace{\qquad}_{u_{sl\_a}(k)} & \end{bmatrix}_{(2n_{obs},2N+n_{obs}+2)} \qquad b_{sl\_p} = \begin{bmatrix} 1 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}_{(2n_{obs},1)}$$

and

$$A_{sl\_a} = \begin{bmatrix} 0 & \dots & 0 & 0 & \dots & 0 & 1 & 0 \\ \vdots & & \vdots & \vdots & & \vdots & 0 & 1 \\ \vdots & & \vdots & \vdots & & \vdots & -1 & 0 \\ 0 & \dots & 0 & 0 & \dots & 0 & 0 & -1 \end{bmatrix}_{(4,2N+n_{obs}+2)} \qquad b_{sl\_a} = \begin{bmatrix} \Delta v_{sl_{\max}} \\ \Delta v_{sl_{\max}} \\ 0 \\ 0 \end{bmatrix}_{(4,1)}$$

and by calling:

$$A_{sl} = \begin{bmatrix} A_{sl\_p} \\ A_{sl\_a} \end{bmatrix} \qquad b_{sl} = \begin{bmatrix} b_{sl\_p} \\ b_{sl\_a} \end{bmatrix} \tag{5.93}$$

it is possible to obtain the classical representation:

$$A_{sl}\bar{\mathcal{U}}(k) \leq b_{sl} \tag{5.94}$$

Chapter 6

Simulation results

In this chapter, numerical results obtained from simulations based on the integration of the two modules described are presented. Simulations were performed on a personal computer equipped with an Intel Core i7 CPU and 16 GB of RAM. The results are obtained from the developed simulation environment, implementing the Model Predictive Controller and the Global Planner algorithm described in the previous chapters in a Matlab&Simulink environment.
The following sections will first briefly introduce the simulation software developed, followed by the results obtained in different simulation scenarios, combining the Global and Local Planner algorithms. More specifically, three different simulation scenarios have been considered to analyze the behaviour of the system in different conditions of pedestrians moving in the environment.

6.1 Simulation Software

According to the navigation strategy described in Chapter 2, the simulation software developed in Matlab&Simulink implements the following modules:

- Model Predictive Controller module
- RRTX (Global Planner) module

The MPC module exploits the Gurobi Optimization software to solve the constrained optimal control problem. In particular, it is possible to define a model structure to solve a minimization problem for a quadratic, convex cost function, with a set of equality and inequality constraints; the optimization problem to

be solved by the Gurobi software can be defined as follows:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & J(x) = x^T H x + f^T x \\ \text{subject to} \quad & A_{ineq} x \leq b_{ineq} \\ & A_{eq} x = b_{eq} \\ & lb \leq x \leq ub \end{aligned}$$

where the matrix H and row vector f constitute the objective function to minimize, while A_{ineq}, b_{inew}, A_{eq}, b_{eq}, lb and ub are matrices and vectors with suitable dimensions, that express the linear inequality constraints, linear equality constraints, lower and upper bounds.
As already described, the vehicle considered is equipped with two Lidar sensors, having a maximum range of 10[m]. In the simulation environment, this can be reproduced including the range of sensors used on the system, as highlighted in figure 6.1. This is done to include, in both the navigation structure layers, only the obstacles that the vehicle is able to detect, and hence reproduce better the navigation condition in a real context.

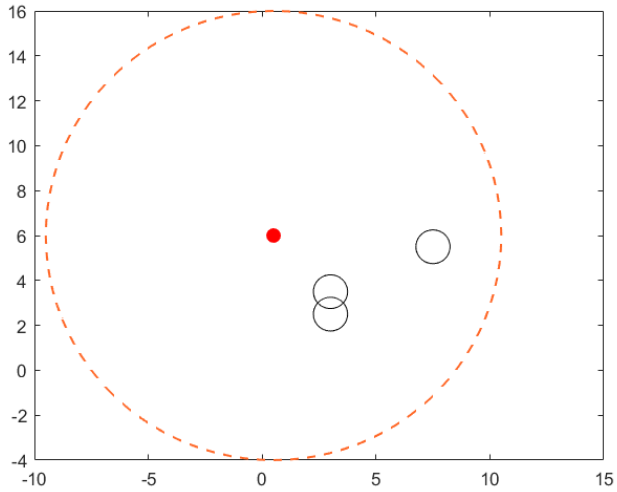


Figure 6.1: Lidar sensor with 10 [m] maximum range, represented by the red dashed line. The red circle represent the vehicle position, while the black bigger circles are the obstacles

6.2 Controller Parameters tuning

Matrices R, Q and S are design parameters that express the penalties given to the control actions and to the error on the state,with respect to the reference, and have to be properly tuned to obtain a suitable control performance. Recalling that the wheelchair is modeled as a symmetric decoupled system, weight matrices can be designed as:

$$Q = \begin{bmatrix} q & 0 \\ 0 & q \end{bmatrix} \qquad R = \begin{bmatrix} r & 0 \\ 0 & r \end{bmatrix} \qquad S = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \tag{6.1}$$

With reference to [12] and [11] for the calibration of r and q weights in simulations, a good trade-off for the MPC parameters was found selecting:

$$\begin{aligned} q = 1 \longrightarrow Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ r = 1 \longrightarrow R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{aligned} \tag{6.2}$$

while the terminal cost s can be retrieved through the relationship:

$$s = \frac{q + k^2 r}{1 - (1 + \tau k)^2} \tag{6.3}$$

The value of k can be obtained exploiting the pole placement method, so that by imposing:

$$k > -\frac{2}{\tau} \tag{6.4}$$

the asymptotic stability of the system is guaranteed, and to avoid an excessive oscillatory behaviour a more conservative condition can be imposed:

$$k > -\frac{1}{\tau} \tag{6.5}$$

Then, a fraction of the limit value can be selected:

$$k = -\frac{1}{n\tau}, \quad n > 1 \tag{6.6}$$

and by selecting $n = 2$:

$$S = \begin{bmatrix} 9.67 & 0 \\ 0 & 9.67 \end{bmatrix} \tag{6.7}$$

The choice of the prediction horizon N, as well as the controller sampling time, depend on the control and computational requirements. To select a proper value

of the prediction horizon N and the controller sampling time τ, the following relationship must hold;

$$N\tau = T_{ph} \tag{6.8}$$

where T_{ph} is the prediction horizon, that is the range of time inside which the controller predicts the system evolution. The controller reactivity increases with a smaller sampling time τ , whereas a high value of N implies a greater computational load. Furthermore, a small value of τ is required to satisfy the hypothesis that, during the controller sampling time, the velocity of each moving obstacle, in particular of pedestrians, remains constant. A good compromise is obtained by selecting T_{ph} equal to $4s$ and:

$$\tau = 0.2s \qquad N = 20 \tag{6.9}$$

Moreover, the choice of T_{ph} must satisfy the following condition:

$$T_{ph} > \frac{v_{max}}{a_{max}} \tag{6.10}$$

that describes the worst case of the breaking time required by the vehicle, so that the vehicle is able to stop within the prediction horizon T_{ph} in every condition. Lastly, recalling the definition of the quadratic cost function:

$$\bar{J}(k) = J(k) + \|u_{sl_p}(k)\|^2_{S_p} + \|u_{sl_a}(k)\|^2_{S_a} \tag{6.11}$$

where:

$$S_p = \begin{bmatrix} s_p & & \\ & \ddots & \\ & & s_p \end{bmatrix}_{(n_{obs},n_{obs})} \qquad s_a = \begin{bmatrix} s_a & \\ & s_a \end{bmatrix}_{(2,2)} \tag{6.12}$$

as described in the previous section,the weights s_p and s_a have to be selected to be much higher with respect to r and q, so that the slack variables will be minimized, if possible, in the cost function. A suitable choice to have good control performance was found as:

$$s_p = 10^9 \qquad s_a = 10^3 \tag{6.13}$$

6.3 Pedestrian Avoidance Simulations

In the following sections the results obtained from simulations performed on the Matlab&Simulink environment are presented, hence proving the consistency of the position constraints, velocity and acceleration of the autonomous wheelchair. Moreover, the results presented allow to show the cooperation of the two navigation layers previously presented. In the simulation results reported in this chapter, the following symbols have been used:

- green circles represent the start and goal positions for the vehicle;
- the black circles and the red ellipses highlight the pedestrian virtual box and personal space function;
- the blue line represents the reference trajectory;
- green stars represent subsequent vehicle positions during motion;
- the red circle is the vehicle's center position (x_C, y_C) while the black circle in front is its linearized position (x_P, y_P);
- black lines display the position constraints at the considered time instant.

As described in previous chapters, a re-planning of the vehicle trajectory happens whenever the tracking error becomes higher than a predefined maximum value ε_{max}, that should be properly selected. Hence, in this section results derived for different values of such limit error value are presented, so to compare different solutions for the cooperation of the two navigation layers.

6.3.1 Simulation 1: Pedestrians and wheelchair moving in the same direction, $\varepsilon_{max} = 1m$

The first simulation considers the motion of the wheelchair along a hallway, taking into account three pedestrians moving in the same direction of the vehicle and starting in front of it. Inputs of the simulation are reported in figure 6.2. This scenario enables to analyse the capability of the vehicle to overtake slower obstacles. Moreover, for this simulation, the maximum tracking error value is set to $\varepsilon_{max} = 1m$

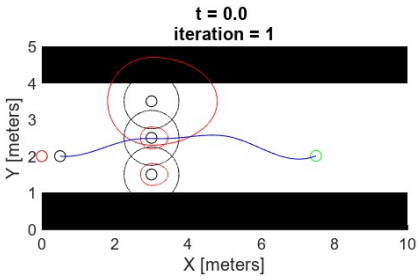
Pedestrians			
	Pedestrian 1	**Pedestrian 2**	**Pedestrian 3**
x_{ped} [m]	3	3	3
y_{ped} [m]	1.5	2.5	3.5
v_{pedx} [m/s]	0.1	0.1	0.4
v_{pedy} [m/s]	0	0	0

Wheelchair		
	Start	**Goal**
x_P [m]	0.5	7
y_P [m]	2	2

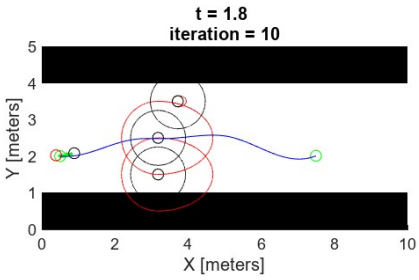
Figure 6.2: Simulation 1 input

Figure 6.3 shows the start wheelchair motion along the planned trajectory while trying to avoid pedestrians moving in front. Position constraints generated

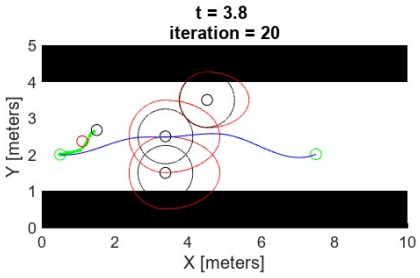
during motion indeed limit the collision-free space for the vehicle, that then needs to deviate from the planned trajectory to avoid collisions with the moving obstacles. The effects of the position constraints on the vehicle motion are depicted in figure 6.4.



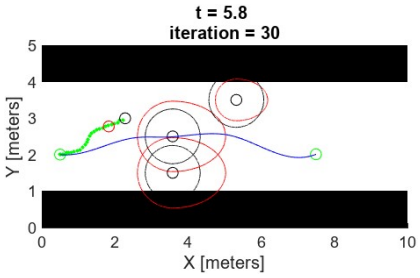
(a) t=0 s



(b) t=1.8 s

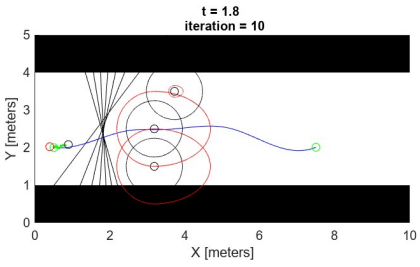


(c) t=3.8 s

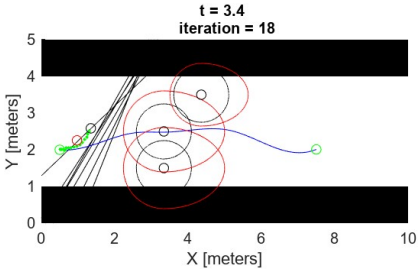


(d) t=5.8 s

Figure 6.3: Simulation 1: Wheelchair motion until re-planning



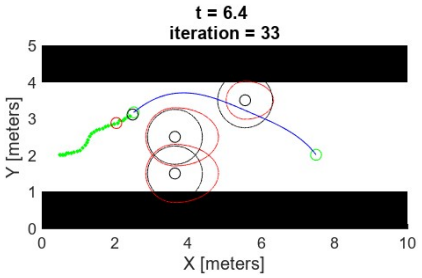
(a) t=1.8 s



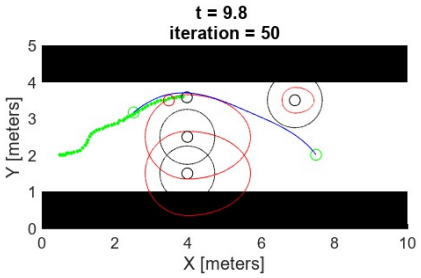
(b) t=3.4 s

Figure 6.4: Simulation 1: Consistency of position constraints

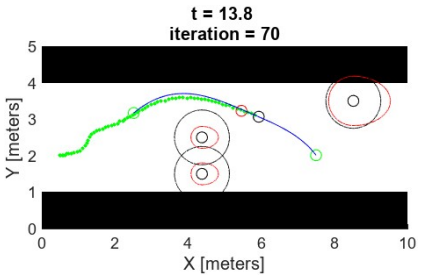
As the vehicle tries to overtake obstacles in front of it, the tracking error ε_{tr} increases. Figure 6.5, then shows the re-planning of a new optimal trajectory performed by the Global Planner, from the vehicle current position to the goal, that is a collision-free trajectory obtained from the path generated by the RRTx algorithm, considering the changes in the environment and a prediction of pedestrians motion within a specified time window. As already explained in chapter 3, the current wheelchair position might not coincide with one of the nodes in the graph. Hence, the starting position for the new optimal trajectory is the node in the graph G closer to the wheelchair. This allows the vehicle to correctly reach its final destination, satisfying all the control requirements and avoiding collisions with all the pedestrians in the environment.



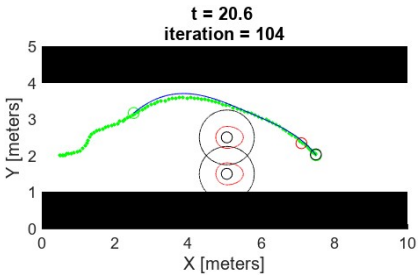
(a) t=6.4 s



(b) t=9.8 s



(c) t=13.8 s



(d) t=20.6 s

Figure 6.5: Simulation 1: Re-plan of the optimal trajectory

Figure 6.6 shows that the distance between the wheelchair center and the pedestrians current positions never exceeds the minimum safety distance required for each obstacle, that is defined as the radius $r_c = 0.75[m]$ of the pedestrian virtual box. Finally, figure 6.7 shows the vehicle velocity profile, together with the linear velocities and the rotational velocities of the wheels, showing the consistency of the linear velocity constraints implemented.

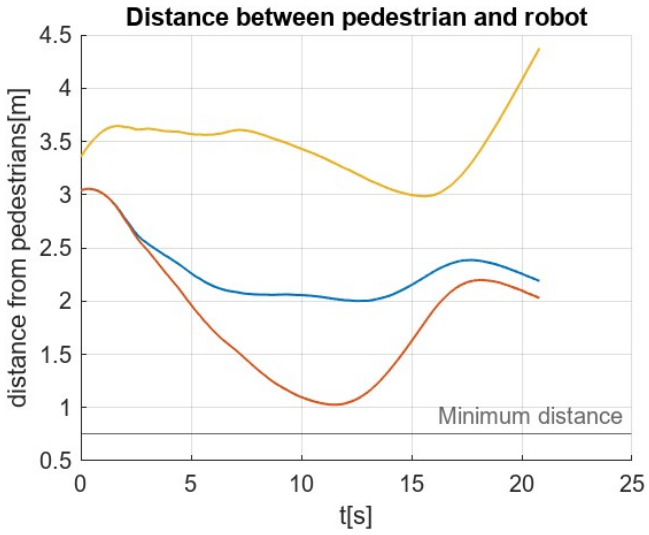


Figure 6.6: Simulation 1: Distance from pedestrians. The black horizontal line represents the pedestrians virtual box radius.

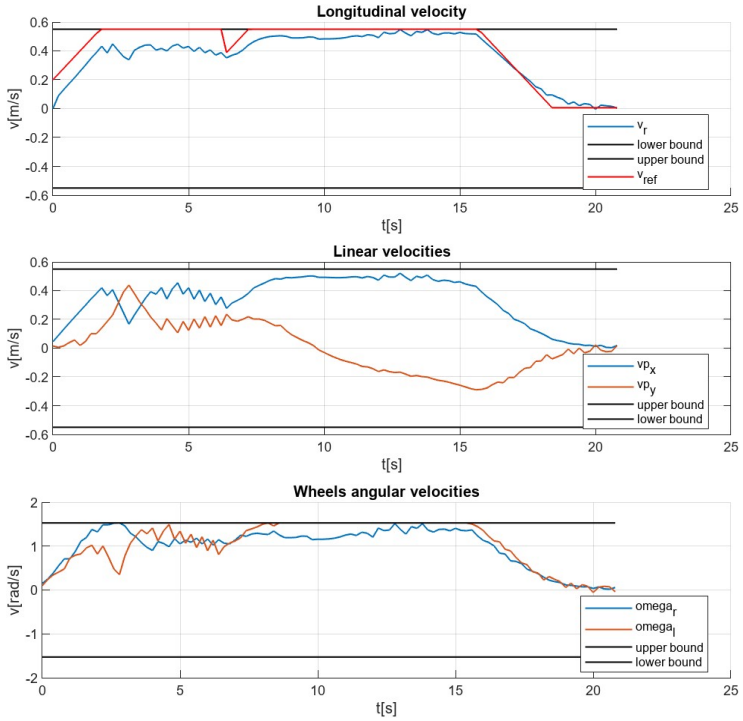
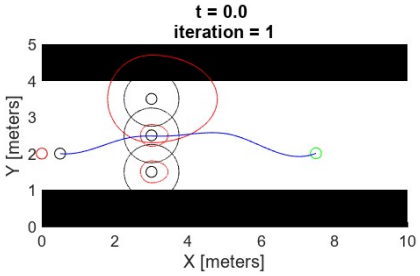


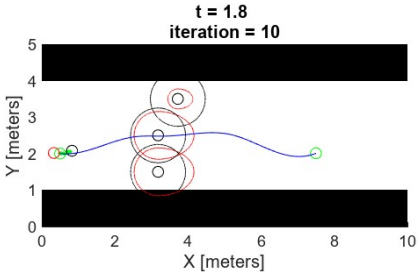
Figure 6.7: Simulation 1: Velocity profiles

6.3.2 Simulation 1: Pedestrians and wheelchair moving in the same direction, $\varepsilon_{max} = 0.5m$

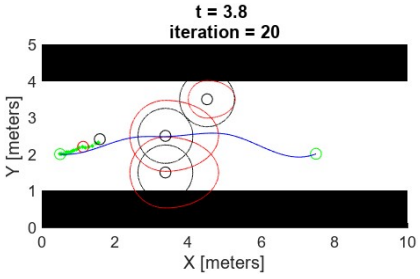
As a comparison, the same simulations has been performed reducing the maximum tracking error to $\varepsilon_{max} = 0.5m$.
In the following, figure 6.8 shows the start of the vehicle motion, until a re-planning action is required. Once the maximum tracking error is reached a new optimal and collision-free trajectory is planned, as shown in figure 6.9. Due to the reduced value of maximum tracking error, with respect to the simulation in6.3.1, the rep-planning action starts at a lower time instant, and then the vehicle can start following the new optimal, collision-free trajectory.
Then 6.10 shows that the vehicle correctly avoids each pedestrian, so that the distance between each pedestrian and the wheelchair is never lower than the minimum distance allowed.
Ultimately, 6.11 shows the velocity profile, the linear velocities and the wheelchair angular velocities of the vehicle. Then, also in this case, angular velocities are considered to show the consistency of velocity constraints.



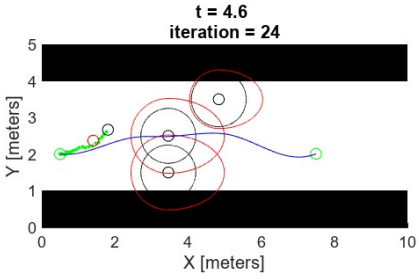
(a) t=0 s



(b) t=1.8 s

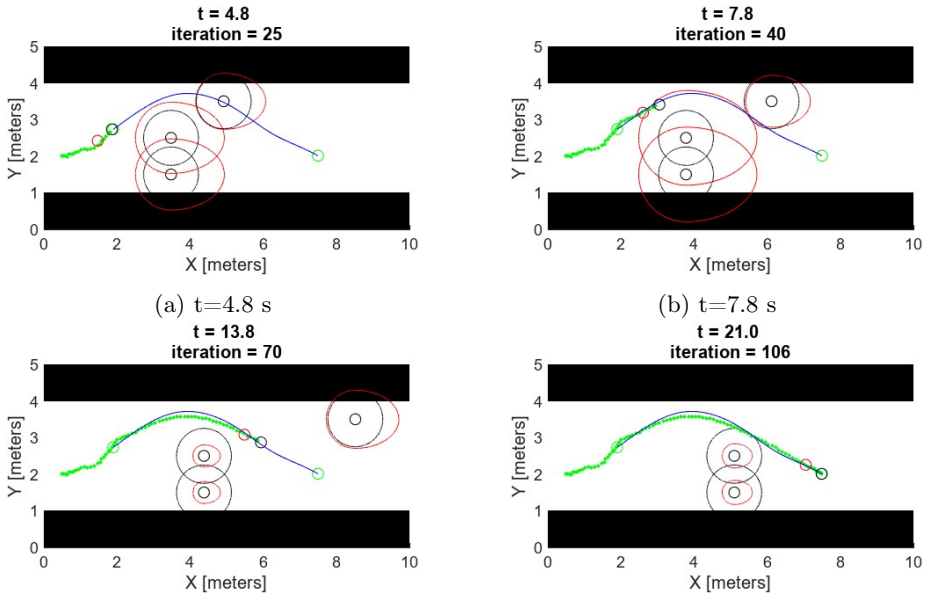


(c) t=3.8 s



(d) t=4.6 s

Figure 6.8: Simulation 1 reduced ε_{max}: Wheelchair motion until re-planning



(a) t=4.8 s

(b) t=7.8 s

(c) t=13.8 s

(d) t=21 s

Figure 6.9: Simulation 1 reduced ε_{max}: Re-plan of the optimal trajectory

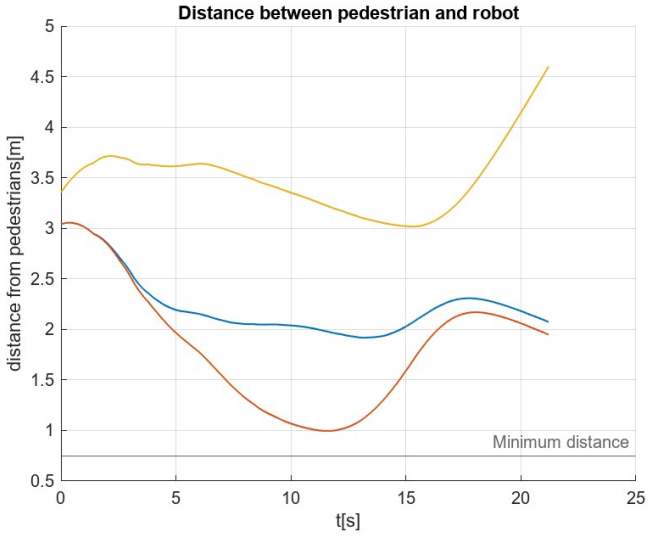


Figure 6.10: Simulation 1 reduced ε_{max}: Distance from pedestrians. The black horizontal line represents the pedestrians virtual box radius.

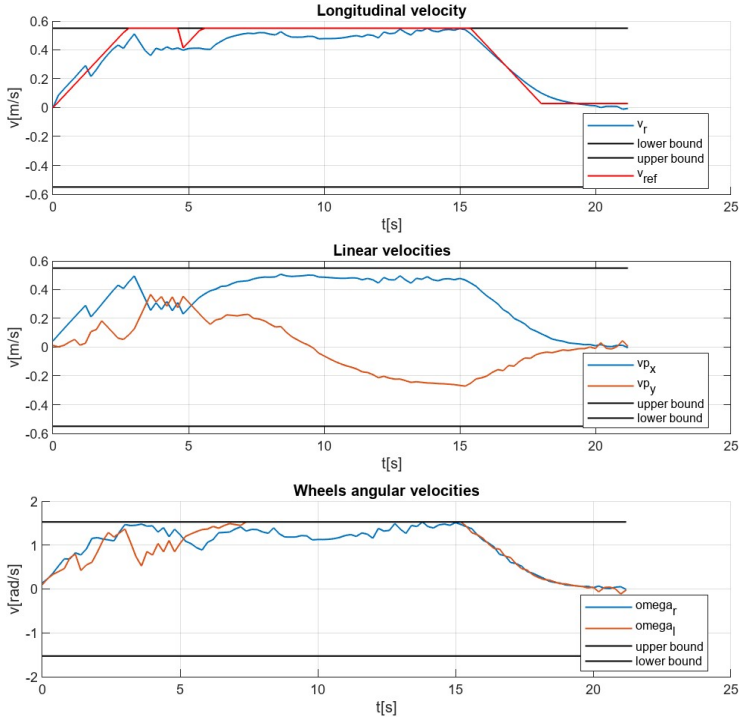


Figure 6.11: Simulation 1 reduced ε_{max}: Velocity profiles

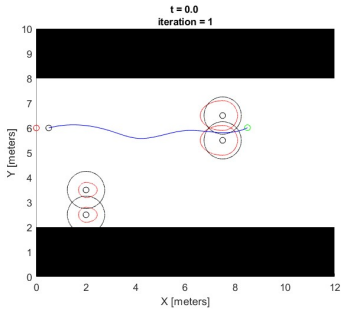
6.3.3 Simulation 2: Pedestrians and wheelchair moving in opposite directions, $\varepsilon_{max} = 1m$

This simulations takes into account four moving pedestrians, where now two of them are moving towards the wheelchair. Inputs of the simulation are reported in figure 6.12. In this simulation setup, the pedestrians moving towards the vehicle have an higher velocity with respect to the other two, hence the wheelchair needs to react to avoid the incoming obstacles. Figure 6.13 shows the evolution of the vehicle in the environment, until a re-planning action is required. Moreover, from figure 6.14 it is possible to see how position constraints generated by the two pairs of pedestrians make the wheelchair deviate from the planned trajectory, so that the future collision can be avoided. In figure 6.15 the re-planning action is displayed, and the new optimal trajectory generated allows the vehicle to reach the goal avoiding the pedestrians, without exceeding the minimum safety distance; this can be seen from figure 6.16, in which the distance between the wheelchair center and the position of each pedestrian is always higher than the virtual box radius r_c.
Ultimately, figure 6.17 shows the velocity profile of the vehicle, the linear velocities and the rotational velocities of the wheels.

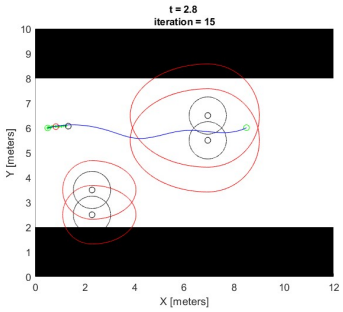
Pedestrians				
	Pedestrian 1	**Pedestrian 2**	**Pedestrian 3**	**Pedestrian 4**
x_{ped} [m]	2	2	7.5	7.5
y_{ped} [m]	2.5	3.5	5.5	6.5
v_{pedx} [m/s]	0.1	0.1	-0.2	-0.2
v_{pedy} [m/s]	0	0	0	0

Wheelchair		
	Start	**Goal**
x_P [m]	0.5	8.5
y_P [m]	6	6

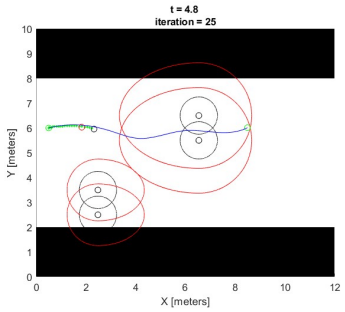
Figure 6.12: Simulation 2 input



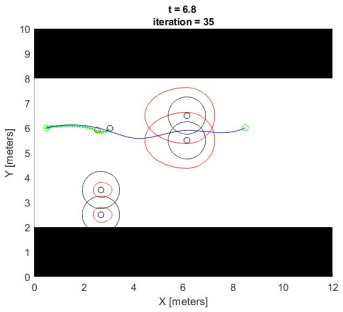
(a) t=0 s



(b) t=2.8 s

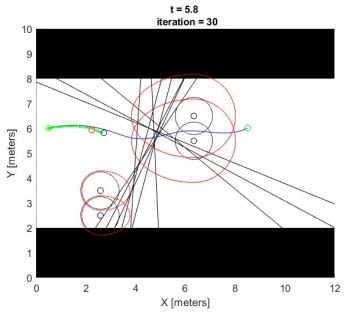


(c) t=4.8 s

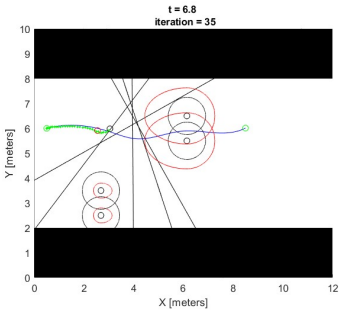


(d) t=6.8 s

Figure 6.13: Simulation 2: Wheelchair motion until re-planning

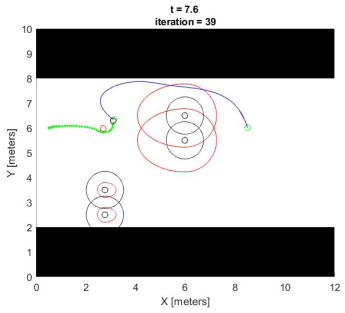


(a) t=5.8 s

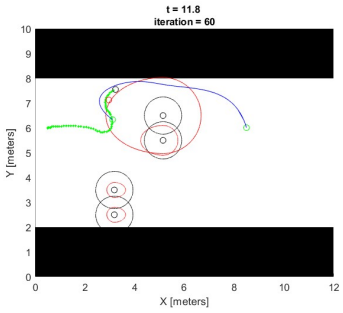


(b) t=6.8 s

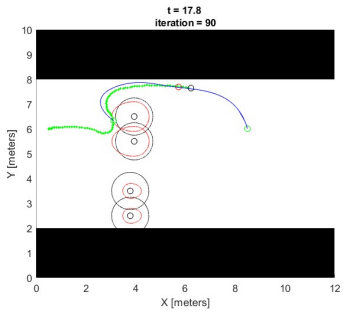
Figure 6.14: Simulation 2: Consistency of position constraints



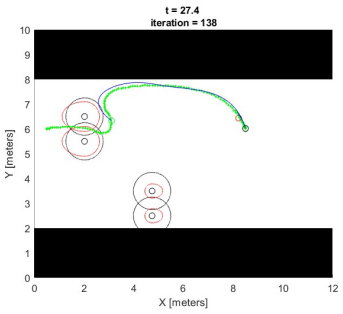
(a) t=7.6 s



(b) t=11.8 s



(c) t=17.8 s



(d) t=27.4 s

Figure 6.15: Simulation 2: Re-plan of the optimal trajectory

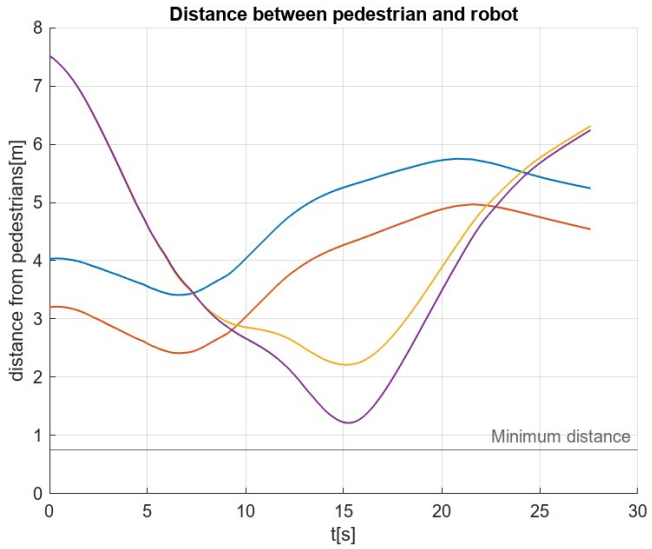


Figure 6.16: Simulation 2: Distance from pedestrians. The black horizontal line represents the pedestrians virtual box radius.

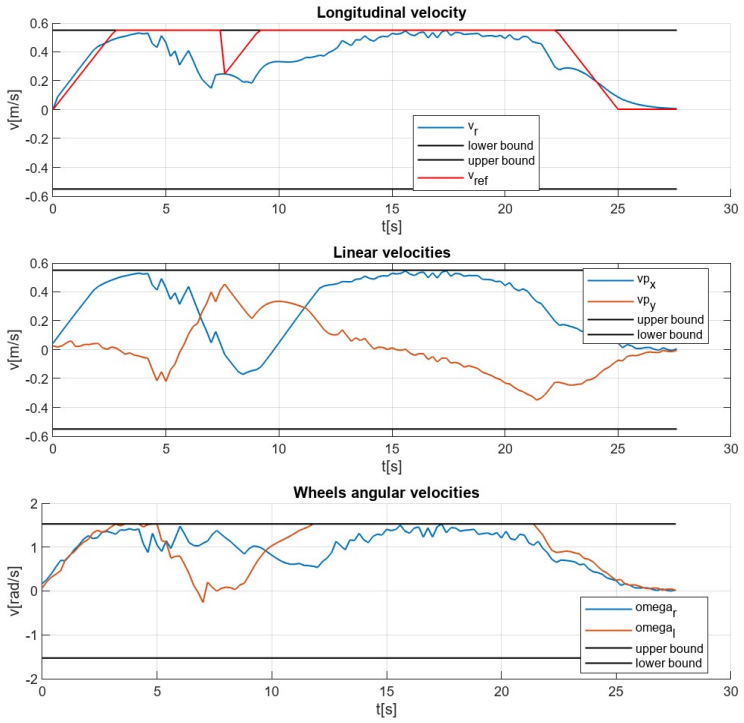
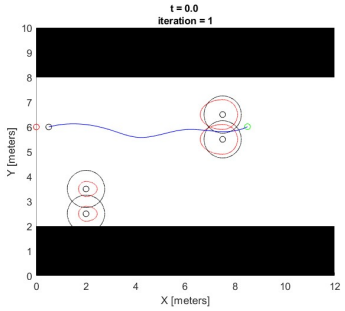


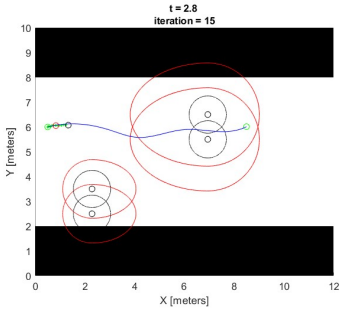
Figure 6.17: Simulation 2: Velocity profiles

6.3.4 Simulation 2: Pedestrians and wheelchair moving in opposite directions, $\varepsilon_{max} = 0.5m$

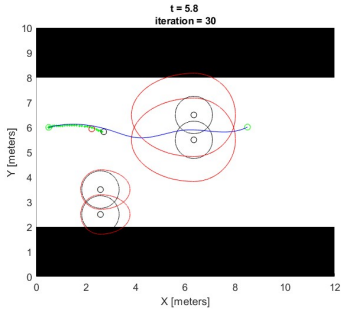
The maximum tracking error is now reduced to $\varepsilon_{max} = 0.5m$ and the same simulation has been performed for this reduced limit value. In figure 6.18 it is possible to see the motion of the vehicle until the re-planning action is required. Then, figure 6.19 shows the re-plan of the optimal trajectory. As the re-planning action is trigger at a lower time instant, with respect to the case displayed in section 6.3.3, pedestrians approaching the vehicle are further from it; hence, a smoother avoidance maneuver is performed through the re-planning of the new path. The vehicle now can follow the collision-free trajectory to its destination position, avoiding collisions with all the pedestrians in the environment.
As in the previous results reported in this chapter, figure 6.20 shows the distance between the vehicle and each pedestrian in the environment, together with the minimum distance required to avoid collisions. In figure 6.21 the velocity profile, linear velocities and wheels rotational velocities of the vehicle are displayed, showing again the consistency of linear velocity constraints.



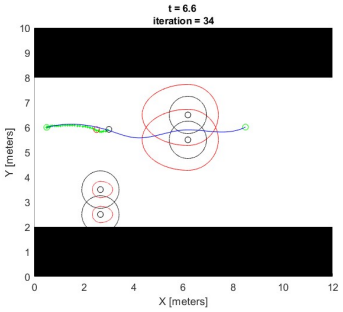
(a) t=0 s



(b) t=2.8 s

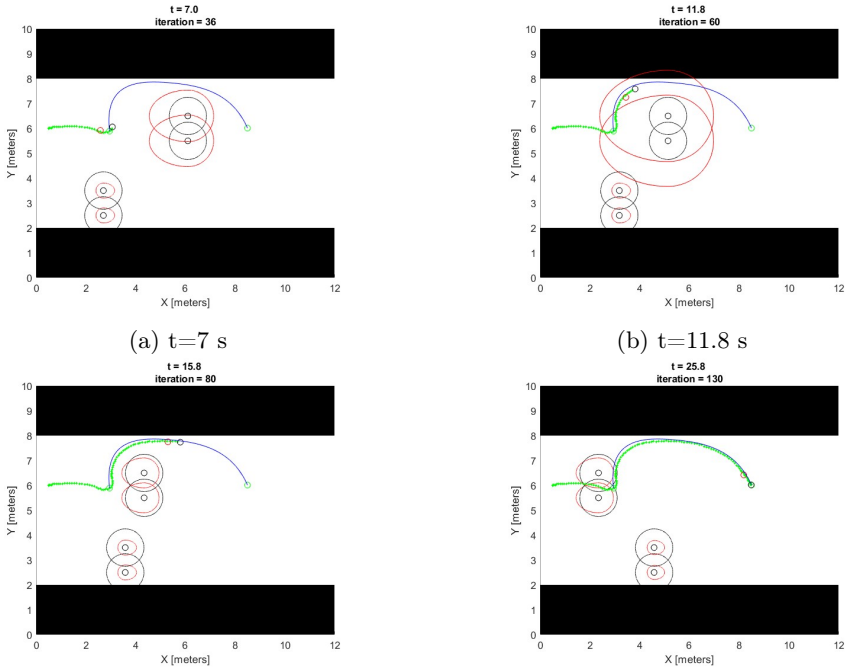


(c) t=5.8 s



(d) t=6.6 s

Figure 6.18: Simulation 2, reduced ε_{max}: Wheelchair motion until re-planning



(a) t=7 s

(b) t=11.8 s

(c) t=15.8 s

(d) t=25.8 s

Figure 6.19: Simulation 2, reduced ε_{max}: re-plan of the optimal trajectory

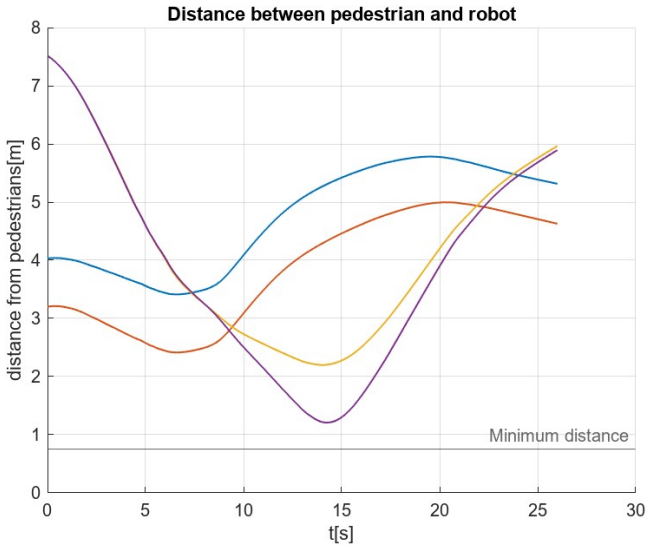


Figure 6.20: Simulation 2 reduced ε_{max}: Distance from pedestrians. The black horizontal line represents the pedestrians virtual box radius.

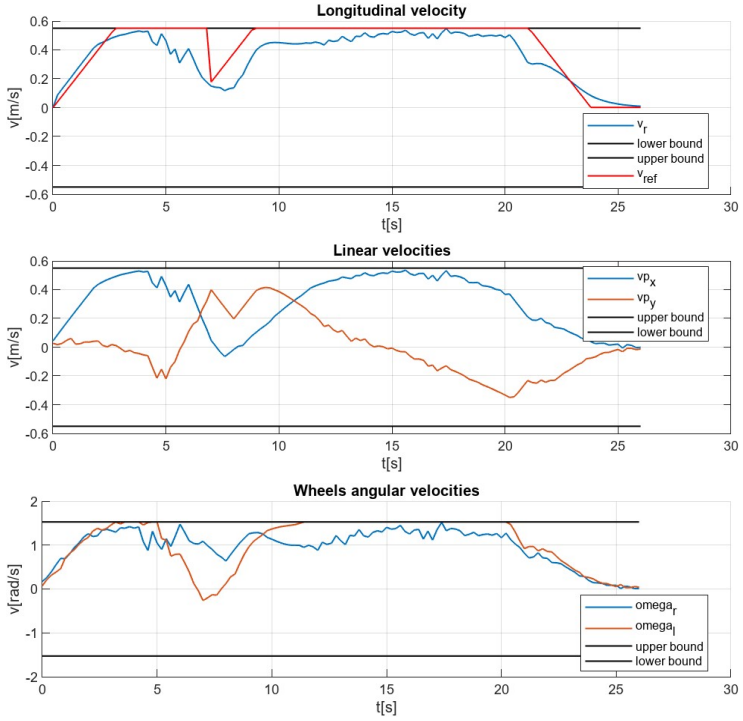


Figure 6.21: Simulation 2 reduced ε_{max}: Velocity profiles

6.3.5 Simulation 3: Motion in a crowded environment, $\varepsilon_{max} = 1m$

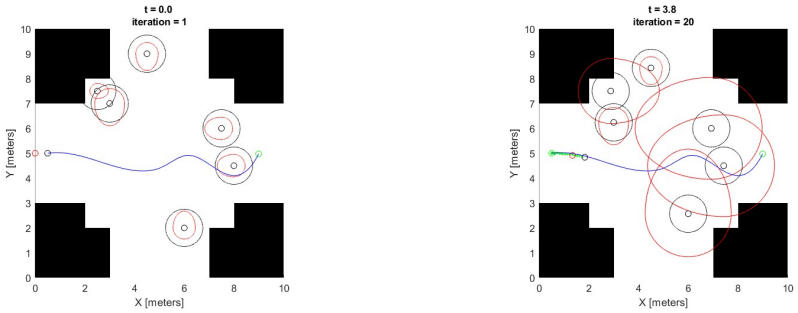
The simulation now presented takes into account the motion of six pedestrians in the environment. Inputs of the simulation are reported in figure 6.22. This scenario allows to analyze the cooperation of the two navigation layers, when the vehicle moves in a dense environment.

Pedoni						
	Pedone 1	**Pedone 2**	**Pedone 3**	**Pedone 4**	**Pedone 5**	**Pedone 6**
x_{ped} [m]	2.5	3	4.5	7.5	8	6
y_{ped} [m]	7.5	7	9	6	4.5	2
v_{pedx} [m/s]	0.1	0	0	-0.15	-0.15	0
v_{pedy} [m/s]	0	-0.2	-0.15	0	0	0.2

Carrozzina		
	Start	**Goal**
x_P [m]	0.5	9
y_P [m]	5	5

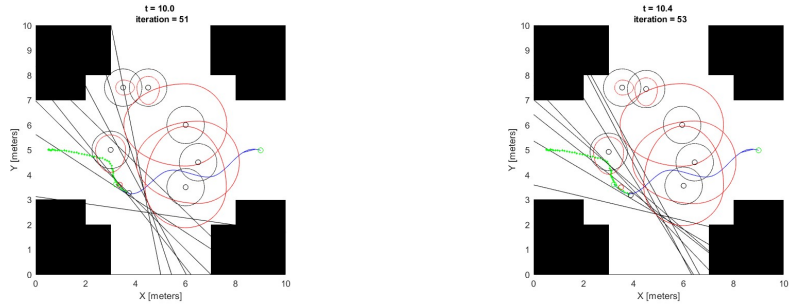
Figure 6.22: Simulation 3 input

In figure 6.23 the beginning of the vehicle motion is displayed, as it starts to follow the planned trajectory. Then, to avoid approaching pedestrians, it needs to deviate from the desired path. The tracking error becomes higher than the maximum value allowed and a first re-planning action is hence triggered, as shown in figure 6.25. However, the socially compliant position constraints (figure 6.24) generated by the controller force the vehicle to keep an higher distance from pedestrians, so that it has to deviate from the planned trajectory. Moreover, a second re-planning of the optimal trajectory is needed, as shown in figure 6.26. The vehicle can now move towards the desired goal, following the new generated reference, as is displayed in figure 6.27.



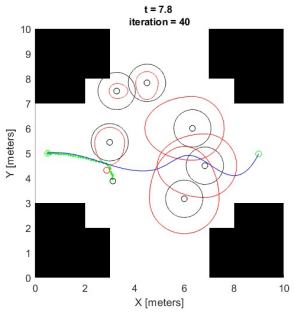
(a) t=0 s (b) t=3.8 s

Figure 6.23: Simulation 3: Wheelchair motion until first re-planning

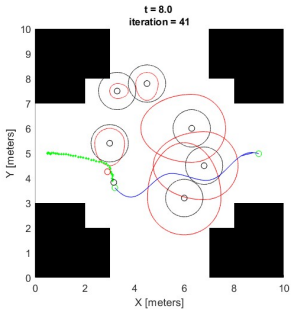


(a) t=10 s (b) t=10.4 s

Figure 6.24: Simulation 3: Consistency of position constraints

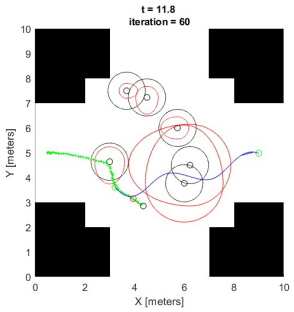


(a) t=7.8 s

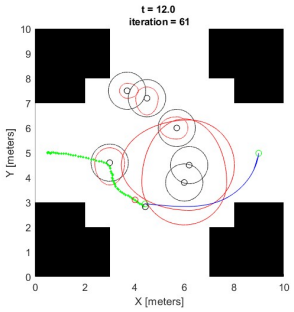


(b) t=8 s

Figure 6.25: Simulation 3: First re-planning action

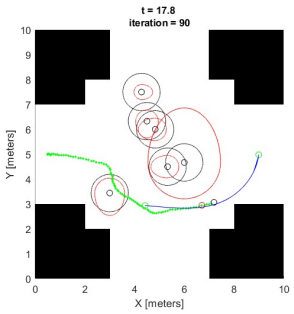


(a) t=11.8 s

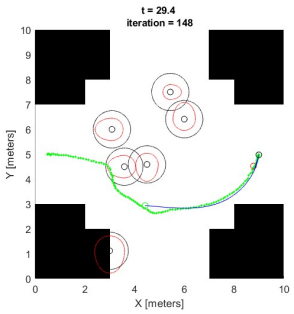


(b) t=12 s

Figure 6.26: Simulation 3: Second re-planning action



(a) t=17.8 s



(b) t=29.4 s

Figure 6.27: Simulation 3: End of motion

Figure 6.28 shows the distance between the vehicle and each pedestrian in the environment, together with the minimum distance required to avoid collisions.

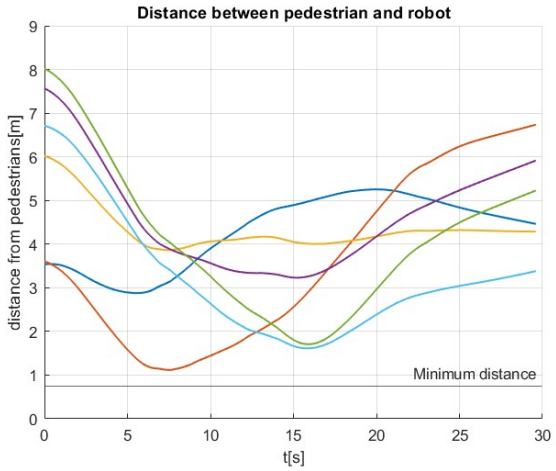


Figure 6.28: Simulation 3: Distance from pedestrians. The black horizontal line represents the pedestrians virtual box radius.

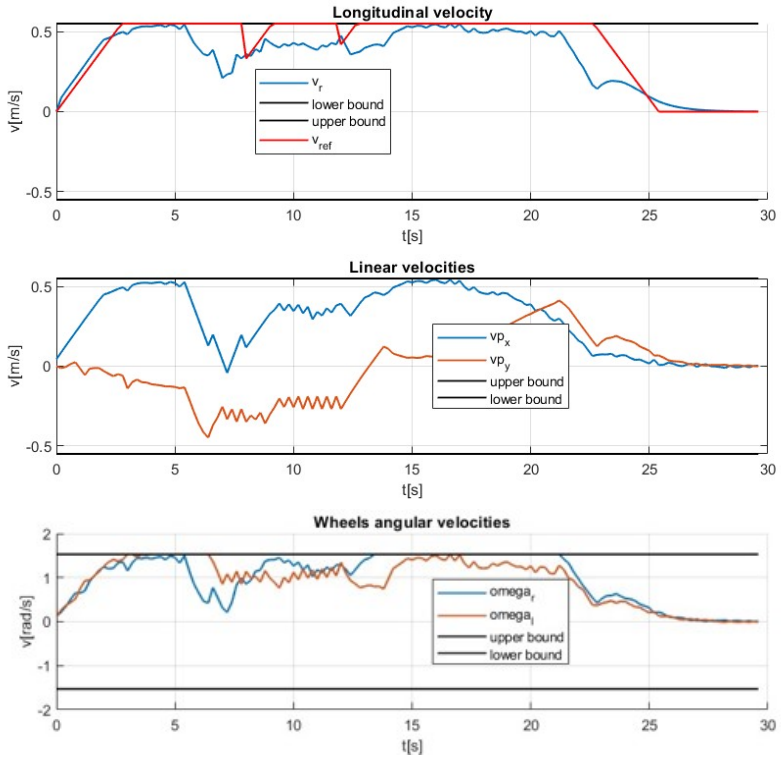
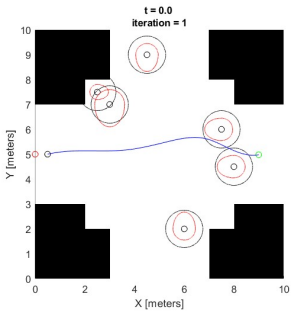


Figure 6.29: Simulation 3: Velocity profiles

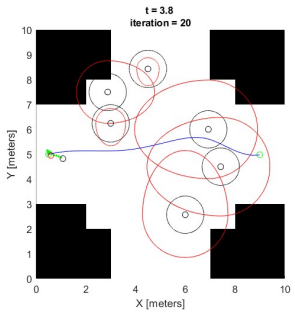
In figure 6.29 the velocity profile, linear velocities and wheels rotational velocities of the vehicle are displayed. The velocity profile also points out the two re-planning time instants previously described.

6.3.6 Simulation 3: Motion in a crowded environment, $\varepsilon_{max} = 2m$

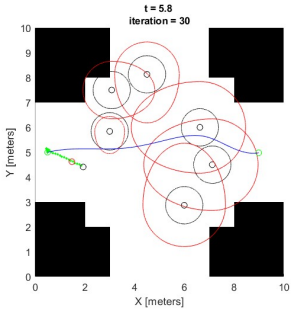
In the simulation scenario presented in 6.3.5, two re-planning action have been required to properly reach the goal position while avoiding collisions with all the obstacles in the environment. The same conditions are now considered with an increased maximum tracking error $\varepsilon_{max} = 2m$. Figure 6.30 shows the start of the vehicle motion, that is not able follow the desired trajectory to avoid collisions with pedestrians. Then, figure 6.31 shows the re-plan of the optimal reference for the vehicle, that can now be tracked and the wheelchair is capable of reaching the desired goal.



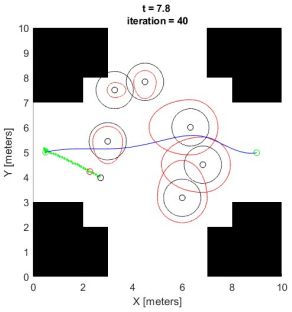
(a) t=0 s



(b) t=3.8 s

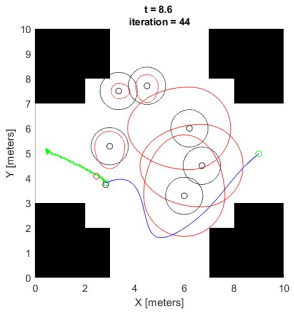


(c) t=5.8 s

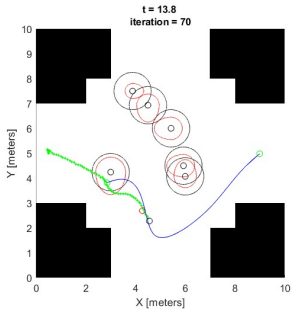


(d) t=7.8 s

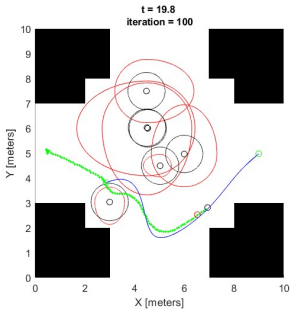
Figure 6.30: Simulation 3 increased ε_{max}: Wheelchair motion until re-planning



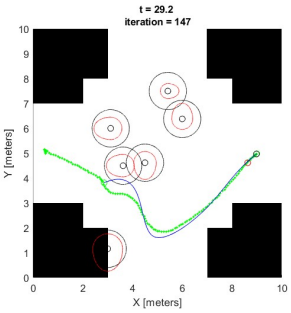
(a) t=8.6 s



(b) t=13.8 s



(c) t=19.8 s



(d) t=29.2 s

Figure 6.31: Simulation 3 increased ε_{max}: re-plan of the optimal trajectory

It is hence clear how, in this case, increasing the maximum tracking error ε_{max} allows the vehicle to reach the destination point with only one re-planning action.
As in all the previous results presented, figure 6.32 shows the distance between the vehicle and all the pedestrians moving in the environment, as well as the minimum distance required to avoid collisions.
Lastly, figure 6.33 shows the velocity profile, linear velocities and the wheels angular velocities of the vehicle, proving the consistency of the implemented velocity constraints.

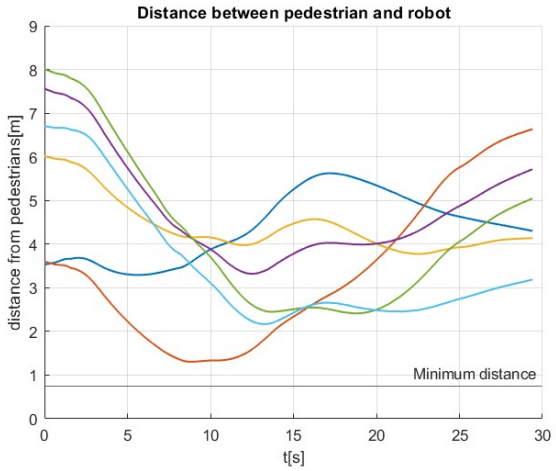


Figure 6.32: Simulation 3 increased ε_{max}: Distance from pedestrians. The black horizontal line represents the pedestrians virtual box radius.

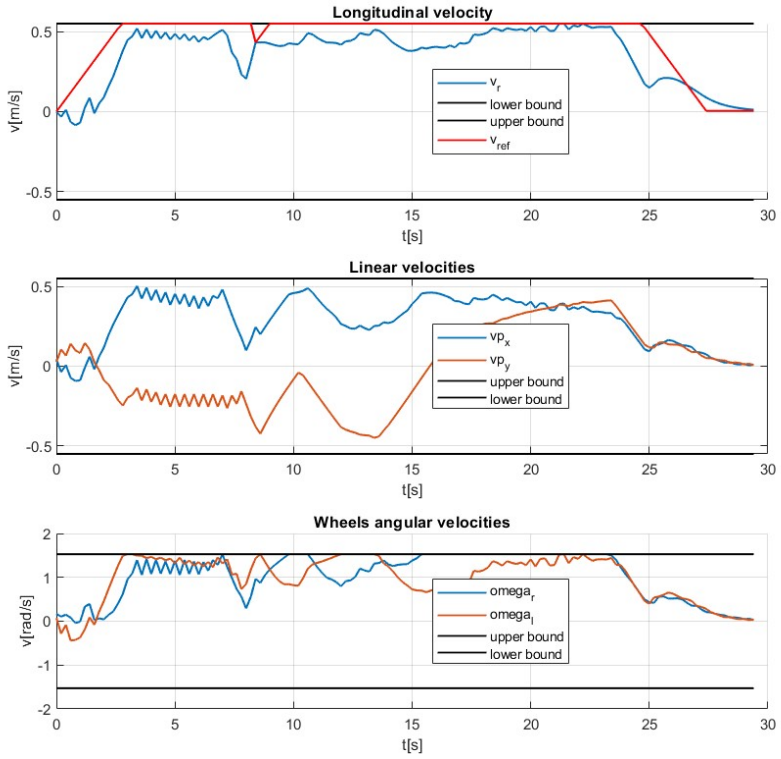


Figure 6.33: Simulation 3 increased ε_{max}: Velocity profiles

Chapter 7

Conclusions

In the work developed in this thesis, a solution for a complete navigation architecture for an autonomous wheelchair has been presented. In particular, the navigation strategy adopted is based on the cooperation of two interlinked layers: the Global Planner and the Controller. The Global Planner computes a trajectory based on the current state of the environment, by generating an asymptotically optimal and collision-free path obtained through the RRTx algorithm, including then a time law thorough a Trapezoidal Velocity Profile. The Controller, implemented as a Model Predictive Controller, tries to track the planned reference trajectory by solving a convex optimization problem that includes constraints on the vehicle maximum performance, as well as position constraints that allow to achieve a socially accepted navigation in a human-crowded environment. Hence, the controller can correct the planned trajectory, according to position constraints that are generated through an obstacle avoidance strategy based on the concept of *Personal Space*, that enables to properly include human comfort requirements in the optimization problem formulation.
To achieve a correct cooperation of the two navigation layers, the tracking error ε_{tr} was used, so that whenever a maximum tracking error value is reached, a new reference trajectory for the vehicle is generated, considering the new state of the environment at the current re-planning time instant.
To prove the effectiveness of the solution proposed, some simulation where performed in various scenarios, showing the proper interaction of the two interlinked navigation layers, as well as the consistency of the constraints implemented in the Model Predictive Control problem formulation. Moreover, for each scenario considered, simulations have been performed for different values of the maximum tracking error ε_{max}, to compare more solutions for the cooperation of the two layers.

Bibliography

[1] Edward T. Hall. *The Hidden Dimension*. Anchor Books, 1966.

[2] Paolo Fiorini and Zvi Shiller. "Motion Planning in Dynamic Environments Using Velocity Obstacles". In: *The International Journal of Robotics Research* 17.7 (1998), pp. 760–772.

[3] Steven Lavalle and James Kuffner. "Rapidly-Exploring Random Trees: Progress and Prospects". In: *Algorithmic and computational robotics: New directions* (Jan. 2000).

[4] Giuseppe Oriolo, Andrea De Luca, and Marilena Vendittelli. "WMR control via dynamic feedback linearization: Design, implementation, and experimental validation". In: *Control Systems Technology, IEEE Transactions on* 10 (Dec. 2002), pp. 835–852.

[5] Rachel Kirby, Reid Simmons, and Jodi Forlizzi. "COMPANION: A Constraint-Optimizing Method for Person-Acceptable Navigation". In: (2009), pp. 607–612.

[6] Riccardo Scattolini Luca Magnani. *Advanced and multivariable control*. Pitagora, 2014.

[7] Michael Otte and Emilio Frazzoli. "RRTX: Asymptotically optimal single-query sampling-based motion planning with quick replanning". In: *The International Journal of Robotics Research* 35 (Sept. 2015).

[8] Gianluca Bardaro et al. "MPC-based control architecture of an autonomous wheelchair for indoor environments". In: *Control Engineering Practice* 78 (Sept. 2018), pp. 160–174.

[9] Basak Sakcak and Luca Bascetta. *Safe Motion Planning for a Mobile Robot Navigating in Environments Shared with Humans*. 2022. arXiv: 2206.07498 [cs.RO].

[10] Alessandro Botti. "GMM/GMR human motion models for path planning in human populated environments". In: *Master thesis, Politecnico di Milano* (2020/2021).

[11] Pierpaolo Durante. "HUMAN-AWARE PLANNING AND CONTROL OF AN INDOOR AUTONOMOUS WHEELCHAIR". In: *Master thesis, Politecnico di Milano* (2017/2018).

[12] Silvia Tellarini. "Autonomous navigation in human crowded environments: integrating model predictive control within global planning". In: *Master thesis, Politecnico di Milano* (2017/2018).