

POLITECNICO DI MILANO

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



An obstacle avoidance algorithm based on
trajectory modification for an autonomous
wheelchair

Relatore: Prof. Luca Bascetta

Tesi di Laurea di:
Amir Sayed Ahmed, matricola 921133

Anno Accademico 2019-2020

Dedicato alla mia famiglia

Ringraziamenti

Innanzitutto, vorrei ringraziare il mio relatore, il Prof. Luca Bascetta, per aver reso possibile questo lavoro di tesi. La sua disponibilità, conoscenza e attenzione ai dettagli mi hanno permesso di apprendere molto e di provare, in prima persona, l'esperienza della ricerca scientifica.

Chi mi conosce sa che arrivare in fondo a questo percorso non è stato semplice: la scelta di affrontare il percorso di laurea da studente lavoratore ha rivoluzionato la mia vita, in positivo, nonostante la fatica e lo stress. Tuttavia, il coronamento di questo risultato non sarebbe mai stato possibile senza la presenza costante di persone che mi hanno dato forza e conforto.

Perciò, un caloroso e speciale ringraziamento va a tutte quelle persone che hanno creduto in me fin dall'inizio, che mi hanno sostenuto nei momenti più difficili e mi hanno permesso di liberare la mente nei periodi più stressanti.

Vorrei esprimere la mia più sincera gratitudine ai miei compagni di corso (nonchè amici), con i quali è stato più che piacevole condividere questo percorso.

Ringrazio infinitamente i compagni del Patio: senza di voi, e le notti passate a studiare insieme, questo risultato non sarebbe stato possibile.

Vorrei inoltre ringraziare tutti i miei amici, che mi hanno permesso lo svago tra un esame e l'altro, alleviando significativamente il peso di questo percorso.

Infine, un ringraziamento speciale va alla mia famiglia, che non ha mai smesso di credere in me e mi ha sempre esortato a dare il massimo.

Abstract

The work developed in this thesis is in the field of mobile robotics, with specific reference to Autonomous Personal Mobility Vehicles (APMV). The goal is to develop an innovative solution to the collision avoidance problem with dynamic obstacles in a trajectory tracking context, done by a system that extends on two layers: Global Planner and Local Planner.

The Global Planner is specialized in planning inside a map, finding a feasible geometric path and endowing it with the time information. The Local Planner solves a trajectory tracking problem, relying on a Model Predictive Control strategy, allowing to express as constraints the requirements in terms of safety, human comfort and collision avoidance while minimizing the distance from the reference trajectory provided by the Global Planner. The approach presented in this work aims to allow collision avoidance in the situations in which the manoeuvre to avoid the obstacle would require moving away from the reference trajectory. This has been achieved by defining an algorithm that predicts potential collisions on the reference trajectory and modifies it to overcome the obstacles. Simulation results show the effectiveness of the proposed solution.

Sommario

Il lavoro svolto in questa tesi appartiene al campo della robotica mobile, con specifico riferimento ai Veicoli Autonomi per la Mobilità Personale (APMV). L'obiettivo è quello di proporre una soluzione innovativa al problema della prevenzione delle collisioni con ostacoli dinamici in un contesto di inseguimento di traiettoria, svolto da un sistema che si estende su due livelli: Pianificatore Globale e Pianificatore Locale.

Il Pianificatore Globale si specializza nella pianificazione all'interno della mappa, calcolando un percorso geometrico fattibile e dotandolo delle informazioni temporali. Il Pianificatore Locale si occupa dell'inseguimento della traiettoria adottando una strategia Model Predictive Control al fine di imporre, sotto forma di vincoli, i requisiti in termini di sicurezza, comfort e prevenzione delle collisioni, minimizzando la distanza dalla traiettoria di riferimento fornita dal Pianificatore Globale.

L'approccio presentato in questa tesi ha l'obiettivo di consentire la prevenzione delle collisioni nelle situazioni in cui la manovra richiesta per evitare l'ostacolo comporterebbe un allontanamento dalla traiettoria di riferimento. Ciò è ottenuto attraverso la definizione di un algoritmo che preveda le eventuali collisioni sulla traiettoria di riferimento e la modifichi opportunamente in modo da permettere il superamento degli ostacoli. I risultati delle simulazioni mostrano l'efficacia della soluzione proposta.

Contents

1	Introduction	1
1.1	Aim of the thesis	1
1.2	Thesis outline	2
2	State of the art	5
2.1	Autonomous Wheelchair	5
2.2	Obstacle Avoidance	6
2.2.1	Repulsive field algorithms	6
2.2.2	Minimum distance algorithms	8
2.2.3	Obstacle yielding algorithms	9
2.3	Obstacle Avoidance with MPC	10
2.4	Pedestrian avoidance behaviour	12
3	Trajectory tracking controller	15
3.1	System Architecture	15
3.2	Problem Definition	18
3.3	Autonomous wheelchair model	21
3.4	Cost function	25
3.5	Velocity constraints	27
3.5.1	Quadratic velocity constraints	27
3.5.2	Linear velocity constraints	28
3.6	Acceleration constraints	29
3.7	Position constraints	31
3.7.1	Socially-compliant constraints	33
3.7.2	Maximally violated line constraints	38
3.8	Conclusions	40
4	Trajectory modification algorithm	41
4.1	Anchor points	41

4.2	Anchor points algorithm definition	57
5	Results	61
5.1	Simulation environment	61
5.2	Model parameters	62
5.3	Quadratic velocity constraints	63
5.4	Linear velocity constraints	73
5.5	Trajectory modification algorithm	74
5.6	Comparison with position constraints	78
5.7	Avoidance in a low density environment	84
6	Conclusions and future work	95
	Bibliography	97

List of Figures

2.1	While moving, the robot increments values where an obstacle is sensed multiple times [1]	7
2.2	Cost function in presence of an obstacle [2]	7
2.3	Example of paths followable in [3]	8
2.4	Example of path followed in [4]	8
2.5	A is the robot, and B is the obstacle. CC is the Collision Cone, containing forbidden relative velocities, which are translated by v_b , defining VO [5]	9
2.6	M is robot position, the two alternatives of accelerating and decelerating are shown in the 3D space. (Passing under the obstacle with respect to t-axis, means reaching the collision point before the obstacle and vice-versa) [6]	10
2.7	This chart shows how MPC evaluates the control sequence for all the prediction horizon, while actually applying just the first control input predicted	10
2.8	The obstacle is represented by the red ellipse, and NMPC predicts its avoidance	11
2.9	When a non-linear obstacle constrain is added, computation time increases	12
3.1	ROS system simplified architecture	16
3.2	Degonda Twist t4 2x2	21
3.3	Laser sensors 360° coverage	21
3.4	Differential drive model, R is wheel radius and d the distance between wheels	22
3.5	Unicycle	23
3.6	Feedback Linearization model	24
3.7	Personal Space function	34

3.8	Contour (a) and surface maps (b) of the two-dimensional asymmetric Gaussian function	35
3.9	Human social interaction space	36
3.10	Position constraint determination (the blue line tangent to \hat{B})	37
3.11	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 defined joining 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.	38
3.12	Case in which the projection on the segment is outside it. .	39
3.13	Case in which the projection on the segment is inside it. .	40
4.1	Initial situation considered, the robot at position $P_r(k)$ is following the the reference trajectory $T_{ref}(k)$ with the velocity $V_r(k)$ and an obstacle is moving along the linear trajectory $l_v^{(j)}(k)$ with velocity $V_o^{(j)}(k)$	43
4.2	The avoidance behaviour can be guaranteed for the robot if it is guaranteed for the point $P_r(k)$ considering the obstacles enlarged by $d_s^{(j)}(k)$	44
4.3	Graphical representation of $d_{\perp}(P, l)$,	47
4.4	Graphical representation of the tangents to $E^{(j)}(k)$ $l_s^{(j)}(k)$ and $l_f^{(j)}(k)$ and the heading line $l_b^{(j)}(k)$ passing through obstacle's back $P_b^{(j)}(k)$	48
4.5	Graphical representation of the anchor point $P_A^{(j)}(k)$ computed for the obstacle j at the discrete time instant k . . .	51
4.6	Graphical representation of the points $P_{A,s}^{(j)}(k)$ and $P_{A,f}^{(j)}(k)$ determined by the intersection of $l_{\perp}^{(j)}(k)$ and the lines $l_s^{(j)}(k)$ and $l_f^{(j)}(k)$	52
4.7	Graphical representation of the point $P_f^{(j)}(k)$, that is the closest one to $P_{A,f}^{(j)}(k)$ belonging to the reference trajectory $T_{ref}(k)$	54
4.8	Graphical representation of the final avoidance trajectory $T_A^{(j)}(k)$	55
4.9	Graphical representation of the merging trajectory $T_l(k)$.	56

LIST OF FIGURES

5.1	Initial situation in which the robot (in blue) is asked to follow the path defined by the two goals represented in green	63
5.2	Actual trajectory followed by the robot to reach first goal, with both quadratic and linear constraints	64
5.3	Actual trajectory followed by the robot to reach the final goal, with both quadratic and linear constraints.	64
5.4	Velocity profile with quadratic constraints. The blue and orange lines represent v_{P_x} and v_{P_y} , the yellow line represents $\ v_P\ $ and the purple one the boundary value v_{max}	65
5.5	Loop time profile with quadratic constraints and <i>BARRIER</i> solver	65
5.6	Actual trajectory followed by the robot to reach the first goal, with only quadratic constraints.	66
5.7	Actual trajectory followed by the robot to reach the final goal, with only quadratic constraints.	66
5.8	Velocity profile with only quadratic constraints.	67
5.9	Loop time profile with only quadratic constraints.	67
5.10	Actual trajectory followed by the robot to reach the first goal, with only linear constraints and <i>BARRIER</i> solver. . .	68
5.11	Actual trajectory followed by the robot to reach the final goal, with only linear constraints and <i>BARRIER</i> solver. . .	69
5.12	Velocity profile with only linear constraints and <i>BARRIER</i> solver	69
5.13	Loop time profile with only linear constraints and <i>BARRIER</i> solver	70
5.14	Actual trajectory followed by the robot to reach the first goal, with only linear constraints and <i>DUAL</i> solver	71
5.15	Actual trajectory followed by the robot to reach the final goal, with only linear constraints and <i>DUAL</i> solver	71
5.16	Velocity profile with only linear constraints. and <i>DUAL</i> solver	72
5.17	Loop time profile with only linear constraints and <i>DUAL</i> solver	72
5.18	Angular velocity profile with linear constraints, the blue and orange line represent ω_L and ω_R while the purple and yellow lines represent the boundary values $\bar{\omega}_m$ and $\bar{\omega}_M$. . .	74
5.19	Initial simple avoidance situation	75

5.20	When in a potential collision, the robot will plan the avoidance trajectory in blue, while considering the magenta trajectory as a reference in next iterations	75
5.21	Robot continues following the avoidance trajectory	76
5.22	The robot will continue to follow the avoidance trajectory until the obstacle is no more in potential collision with the reference one (in magenta)	76
5.23	After avoiding the obstacle, the robot will continue following the merging trajectory	77
5.24	The robot finally reaches the original reference trajectory and continues following it	77
5.25	Distance between robot and obstacle (on y-axis), with respect to time (on x-axis). The orange line represents the boundary value $d_s = 0.5[m]$ while the blue line represents the actual value during the simulation	78
5.26	Initial situation	79
5.27	Situation in which the robot starts detecting the obstacle. The magenta line represents the ΔV line, while the red ones are the constraints. The red circle represent the obstacle and the blue points represent robot positions	79
5.28	The robot continues following the reference trajectory	80
5.29	Robot is not able to properly avoid the obstacle	80
5.30	Detail of the colliding area	81
5.31	Distance between robot and obstacle (on y-axis), with respect to time (on x-axis). The orange line represents the boundary value $d_s = 1[m]$ while the blue line represents the actual value during the simulation	81
5.32	The algorithm builds the first avoidance trajectory and modifies the reference trajectory by inserting the magenta merging trajectory	82
5.33	Robot follows the avoidance trajectory	82
5.34	After overcoming the obstacle, the robot will follow the merging trajectory	83
5.35	After following the merging trajectory, the robot has reached again the original reference trajectory	83

LIST OF FIGURES

5.36	Distance between robot and obstacle (on y-axis), with respect to time (on x-axis). The orange line represents the boundary value $d_s = 1[m]$ while the blue line represents the actual value during the simulation	84
5.37	Initial situation, the robot starts planning for avoiding the first obstacle	85
5.38	Robot is following the avoidance trajectory for the first obstacle	85
5.39	The avoidance maneuver for the first obstacle has been successfully completed	86
5.40	Robot has reached again the original reference trajectory	86
5.41	Situation in which an obstacle potentially collides with the avoidance trajectory of another obstacle	87
5.42	After the second obstacle, the robot will continue avoiding the third one	87
5.43	Final situation in which the robot has reached again the original reference trajectory	88
5.44	Distance between robot and obstacles (on the y-axis), with respect to time (on the x-axis). The purple line represents the boundary value $d_s = 1[m]$ while the blue, yellow and green lines represent the actual value during the simulation	88
5.45	Initial situation with two obstacles coming from opposite sides	89
5.46	The robot plans the avoidance trajectory for the first obstacle	90
5.47	The robot continues following the avoidance trajectory for the first obstacle	90
5.48	The robot has avoided the first obstacle and detects the second one, which has the avoidance trajectory in potential collision with the first obstacle	91
5.49	The first robot is no more in potential collision with the avoidance trajectory, and the robot continues following it for the second obstacle	91
5.50	The robot has avoided also the second obstacle and follows the merging trajectory	92
5.51	The robot has successfully reached again the original reference trajectory	92

5.52 Distance between robot and obstacle (on the y-axis), with respect to time (on the x-axis). The purple line represents the boundary value $d_s = 1[m]$ while the blue and yellow lines represent the actual value during the simulation for the first and second obstacle, respectively 93

Chapter 1

Introduction

The work developed in this thesis focuses on the motion planning of autonomous vehicles, known in the literature as AGVs (*Automated Guided Vehicles*) in the field of mobile robotics. The applications in this field have witnessed a significant increase during the last two decades, across different fields like Automotive, Manufacturing and Healthcare. These robots can be used to solve many tasks in autonomy by reducing, or even removing, human labour.

The design of an AGV requires mainly to address problems related to autonomous navigation, trajectory generation, obstacle detection and avoidance during its motion.

1.1 Aim of the thesis

The work done in this thesis focuses on autonomous navigation in low people density environments, with specific reference to APMV (*Autonomous Personal Mobility Vehicles*). The usage of these vehicles allows to assist people with motor disabilities or neurological diseases like the ALS (*Amyotrophic Lateral Sclerosis*) or Parkinson, which prevent the ability to control the movement of an electric wheelchair autonomously; thus, the usage of an APMV represents a good solution.

In particular, previous thesis works have put the basis for this case study, where the overall system has been structured on two layers: Global Planner and Local Planner.

The Global Planner is the level specialized in planning a feasible trajectory within the environment by choosing a geometric path and endowing it with the time information.

On the other hand, the Local Planner is specialized in control by solving a trajectory tracking problem while detecting obstacles and guaranteeing their avoidance.

In particular, the trajectory tracking problem is formalized as an optimization problem to be solved with a Model Predictive Control strategy while guaranteeing the requirements in terms of comfort and safety.

This thesis aims to provide a novel technique based on the modification of the reference trajectory to provide obstacle avoidance with specific reference to pedestrians.

In that way, it is possible to bring at the Local Planner level the information related to which direction should be followed by the robot to avoid the obstacles it encounters appropriately.

To be specific, the Global Planner will not be considered since it is outside the scope of this work. Therefore, the same Local Planner presented in [7] will be used, with the specific objective to improve the collision avoidance behaviour.

In particular, the uncrowded scenarios, in which the pedestrians could move freely at a higher speed, will be considered. Thus a different approach for the solution of the collision avoidance problem is shown.

Finally, the results obtained from simulations show the effectiveness of the proposed solution in the considered case study.

1.2 Thesis outline

The thesis is structured in the following way:

- Chapter 2 introduces the state of the art of obstacle avoidance in the mobile robotics field, as well as its integration with *Model Predictive Control* for Local Planning.
- Chapter 3 presents the mathematical model representing the autonomous wheelchair and the definition of the *Model Predictive Control* trajectory tracking optimization problem, with the constraints required to guarantee comfort and safety during navigation.
- Chapter 4 presents the proposed strategy for obstacle avoidance and a trajectory modification algorithm allowing its implementation in a trajectory tracking context.

1.2 Thesis outline

- Chapter 5 shows the results obtained in the implementation of different constraints in the system developed in [8], as well as the implementation of the proposed obstacle avoidance strategy in a simulation environment.
- In Chapter 6, some conclusions are carried out followed by the discussion of possible future developments and improvements.

Chapter 2

State of the art

In the last years, in literature interest in autonomous mobile robots has increased, since most of the technologies required to implement a fully autonomous driving robot have been developed and improved. A brief overview of autonomous wheelchairs is given in the following sections, converging on the different approaches adopted to implement the obstacle avoidance behaviour. In the last section, different models for pedestrian avoidance are shown, describing the yielding behaviour, for which implementation a novel approach will be provided in this work.

2.1 Autonomous Wheelchair

Even if autonomous wheelchairs have started spreading more than twenty years ago, still it is hard to find a commercial, complete solution. This one should provide a fully autonomous experience, guaranteeing safety, with obstacle avoidance and velocity constraints. To achieve that, the topics of HMI (Human Machine Interface), Localization, Planning and Obstacle Avoidance must be addressed, providing a reliable solution to each problem. However, due to this work's aim, more focus will be given to the description of Obstacle Avoidance's different implementations. One of the main requirements for an autonomous wheelchair is to give the user the ability to communicate its desired position effectively. As shown in [9], different techniques as electrooculograms (i.e., translating eye movements in a signal), voice recognition, and gestures can be adopted. Although which one should be used heavily depends on the type and level of disability of the end-user. Another requirement is to equip the wheelchair with a reliable detection system, allowing it to correctly sense the surrounding

environment. For example, in [5], an implementation of an autonomous wheelchair, equipped with a laser range-finder and two infrared scanners for short-range sensing is described, showing its ability to navigate in a crowded environment. Another example is [10], where a 3D sensor is used to detect the environment, planning and avoiding obstacles in 3D space, although this type of sensors is strongly affected by lighting conditions.

2.2 Obstacle Avoidance

In literature, different approaches to the obstacle avoidance problem have been carried out. Here, some of them are briefly reported, grouping them based on the main concept on which the algorithm relies:

- obstacles are seen as *repulsive* fields;
- robot keeps a minimum safety distance from any obstacle, moving towards the goal;
- after estimating the obstacle velocity, the robot yields to avoid a potential collision.

2.2.1 Repulsive field algorithms

This class of algorithms implements, either using a cost map or a cost function, a collision-avoidance behaviour where the robot avoids moving toward a position corresponding to a high-cost cell or a high-cost value. One example is CARMEL robot (Computer-Aided Robotics for Maintenance, Emergency, and Life support) [1], which builds in real-time a histogram map (Figure 2.1). This map cells are filled with a value representing the level of evidence for obstacle existence. That is, a coefficient as large as the probability to find an obstacle in a specified cell. After building the map, the robot applies the Virtual Force Field (VFF), a collision avoidance algorithm that steers the robot proportionally to the level of evidence calculated before.

2.2 Obstacle Avoidance

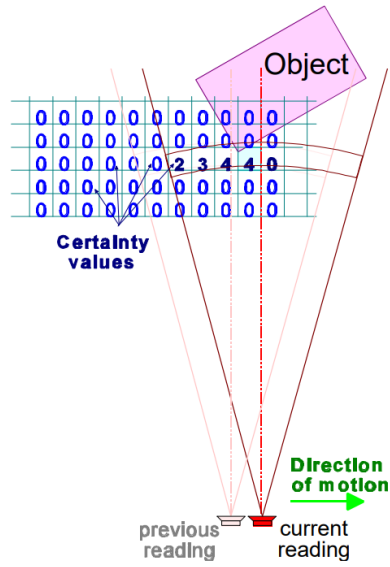


Figure 2.1: While moving, the robot increments values where an obstacle is sensed multiple times [1]

Another example is the work done in [2], where, using Nonlinear Model Predictive Control (NPMC), an obstacle-repulsive cost function is defined (Figure 2.2), allowing the vehicle to steer and avoid collisions correctly.

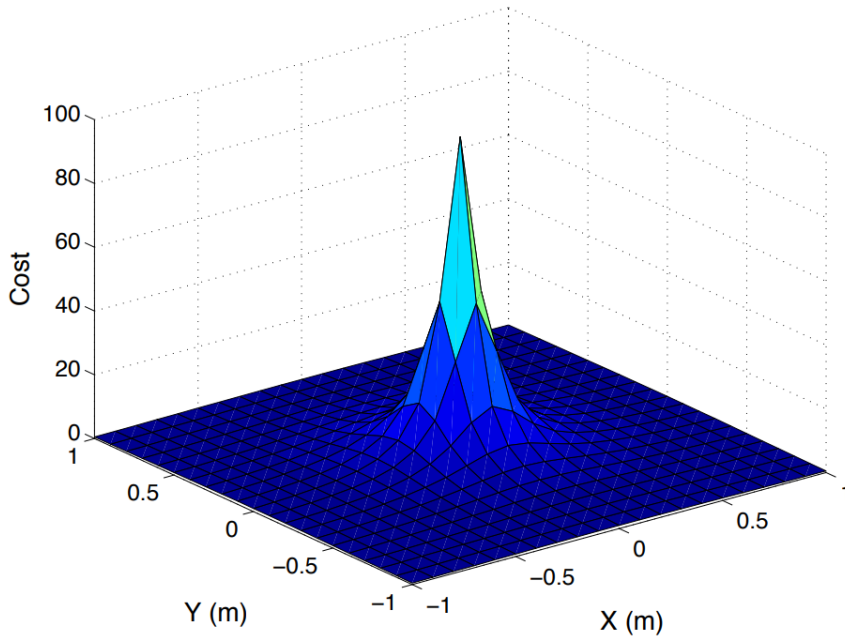


Figure 2.2: Cost function in presence of an obstacle [2]

2.2.2 Minimum distance algorithms

In this type of algorithms, when the robot detects an obstacle, it will always constrain its motion to keep a minimum safety distance from the obstacle, realigning its orientation pointing towards the goal whenever that direction will be no more constrained. This approach has revealed to be more robust in cluttered environments and when dealing with a complex-shaped obstacle. In fact, as shown in [4] and [3], (Figure 2.4 and 2.3, respectively) the robot is able to overcome obstacles following their shape, even if complex.

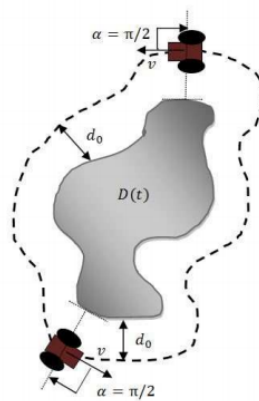


Figure 2.3: Example of paths followable in [3]

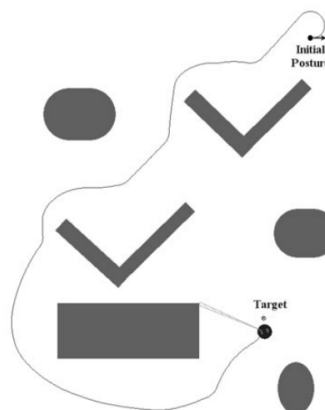


Figure 2.4: Example of path followed in [4]

2.2 Obstacle Avoidance

2.2.3 Obstacle yielding algorithms

This class of algorithms assumes that an obstacle position and velocity are known or estimated from sensor data. Whenever robot and obstacle trajectories may intersect, the robot steers until avoiding the obstacle. This approach has been successfully implemented in [5], using the Velocity Obstacle method, where, evaluating the relative velocity between robot and obstacle, it is possible to define the Velocity Obstacle, i.e., a polygon containing all velocities that would lead to a collision (Figure 2.5), and then evaluate, by difference with the set of reachable velocities, the set of Reachable Avoidance Velocities (RAV), from which the next velocity control to be applied is chosen.

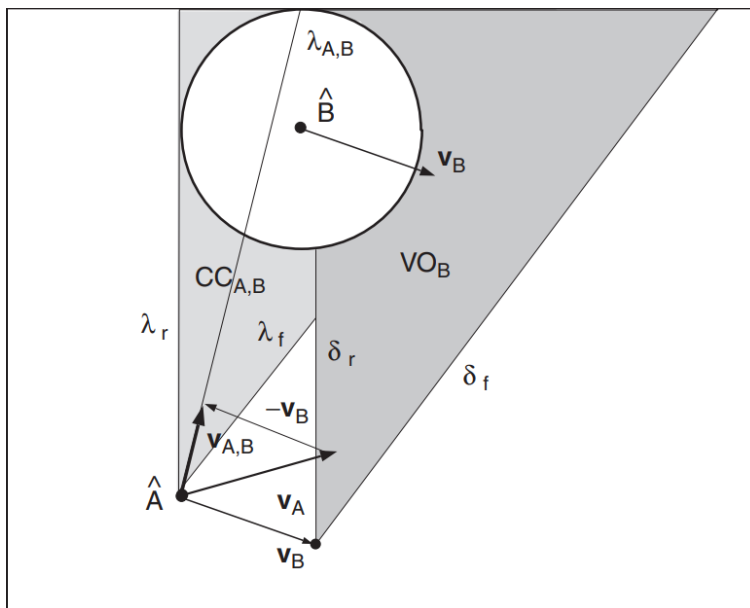


Figure 2.5: A is the robot, and B is the obstacle. CC is the Collision Cone, containing forbidden relative velocities, which are translated by v_b , defining VO [5]

Another similar technique is the one implemented by [6], where obstacle trajectories get projected in a 3D space having time on the z-axis. The robot then plans a trajectory in that space, either accelerating or decelerating, avoiding all obstacles (Figure 2.6).

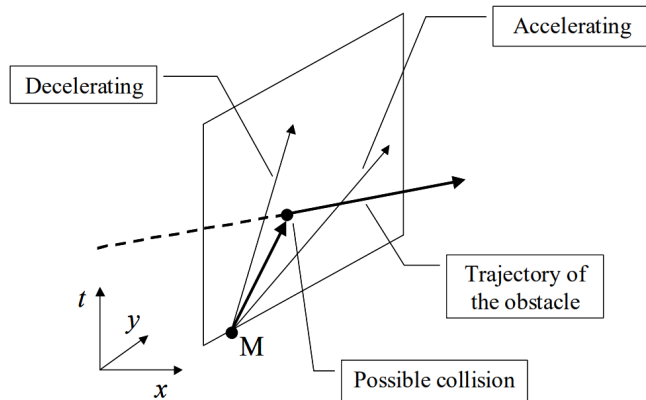


Figure 2.6: M is robot position, the two alternatives of accelerating and decelerating are shown in the 3D space. (Passing under the obstacle with respect to t-axis, means reaching the collision point before the obstacle and vice-versa) [6]

2.3 Obstacle Avoidance with MPC

During the last forty years, Model Predictive Control (MPC), an advanced process control method, has gained increasing popularity. Its primary purpose is (starting from the dynamic system model) to evaluate the next control sequence solving an optimization problem, defined by a cost function and a set of constraints. Then, at each iteration, the optimization problem will be evaluated for a fixed-length time horizon, finding the optimal sequence of the next control inputs, and executing just the first one (see Figure 2.7).

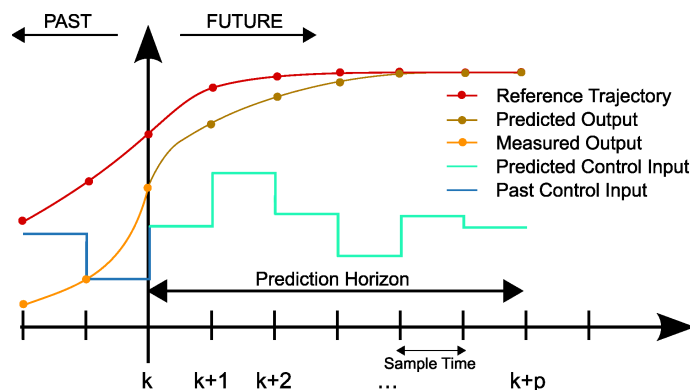


Figure 2.7: This chart shows how MPC evaluates the control sequence for all the prediction horizon, while actually applying just the first control input predicted

2.3 Obstacle Avoidance with MPC

During the years, this control strategy has proven to be efficient and reliable, and many works had also attempted to use it to achieve an obstacle avoidance behaviour. For example, in [2], using Nonlinear Model Predictive Control (NMPC), this has been achieved adding obstacles as soft constraints into the cost function; while in [11], obstacles are seen as non-linear and non-convex constraints (i.e. robot should not enter inside an ellipsis surrounding the obstacle, see Figure 2.8).

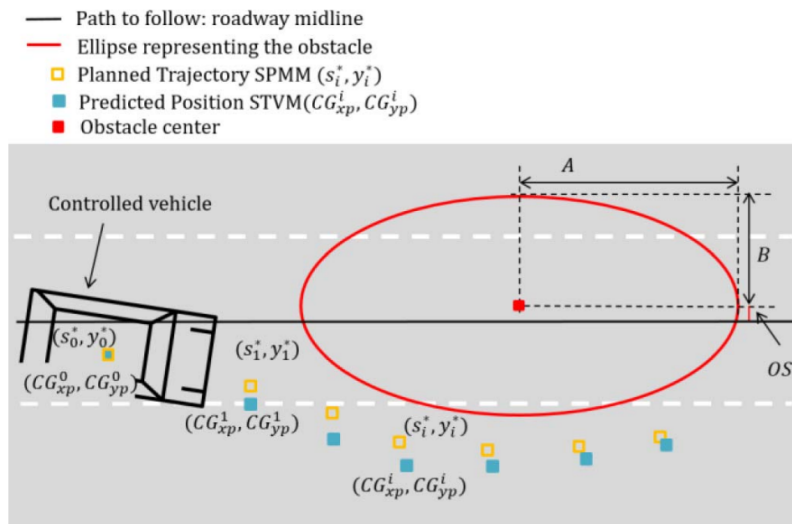


Figure 2.8: The obstacle is represented by the red ellipse, and NMPC predicts its avoidance

One of the main issues related to the use of NPMC is computational time, which may significantly increase when dealing with non-linearities, either in cost function or in constraints. The work done in [2] has shown clearly that effect on a report chart (see Figure 2.9)

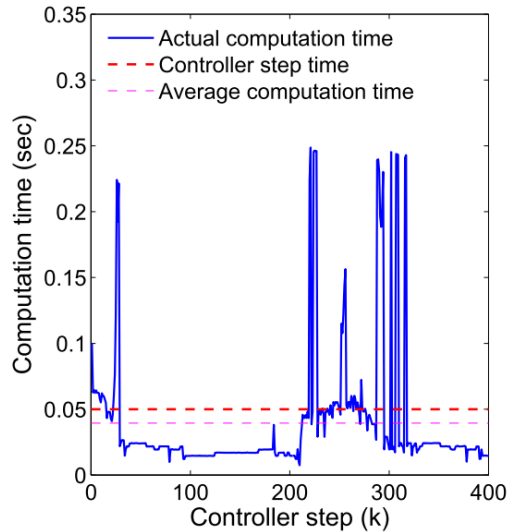


Figure 2.9: When a non-linear obstacle constrain is added, computation time increases

2.4 Pedestrian avoidance behaviour

In recent studies, many attempts to reproduce and analyze human behaviour when it comes to obstacle avoidance have been carried out. In particular, the work done in [12], which consists in an obstacle avoidance technique based on the use of Bayesian Neural Networks, has identified in the avoidance behaviour three sub-tasks (each of them, associated to a different Neural Network):

- passing through a door/corridor;
- wall following;
- general obstacle avoidance

Then the Bayesian framework is exploited to determine which network structure should be used. A good analysis of pedestrian behaviour can be found in [6], where crowds can be modelled from either a macroscopic or a microscopic point of view. The macroscopic approach, which describes crowds with fluid-like properties, is commonly used to evaluate the effect of a modification in an environment design or, if it is possible, to evacuate the structure in time in case of emergency. However, the microscopic approach better describes the interaction between pedestrians, recognizing three different behaviour models:

2.4 Pedestrian avoidance behaviour

- A yielding model, i.e., changing one's trajectory to avoid the collision. Yielding can happen from 1.5m to 30m distance, depending on the environment density.
- A group following model, where each component tends to keep separate from its mates, aligned with their velocity. Although, in reality, each agent behaviour depends on its personality.
- A particle system model, in which crowds are represented as particle clouds. That model is used to study the position of congestion points and spread of panic phenomena.

In this work, a novel approach to implement the yielding behaviour on an autonomous wheelchair will be provided, to be used in a Model Predictive Controller, with only linear constraints, keeping computation complexity of the optimization problem as lower as possible.

Chapter 3

Trajectory tracking controller

In the following sections, starting from the work done in [8], a ROS system for the control of an autonomous wheelchair is introduced. In section 3.2, the MPC problem definition is introduced, followed by the wheelchair model in section 3.3. Finally, in sections 3.5, 3.6 and 3.7 the constraints to be set for fulfilling requirements on velocity, acceleration and position are shown. In particular, in section 3.5, two different sets of velocity constraints are reported; while in section 3.7, the position constraints used to avoid obstacles are described.

3.1 System Architecture

In this work, starting from the developments done in [8] and [13], a ROS (Robot Operating System) system has been used to implement the control of the autonomous wheelchair. ROS is a middleware framework distributed with a set of tools useful for robot systems development, as described in [14]. The main advantage brought by ROS is that it allows the definitions of node-oriented architectures, while handling the communication and synchronization complexity, providing the ability to communicate between two nodes, both synchronously or asynchronously. Synchronous behaviour is implemented by defining a *ROS Service*, which other nodes can call. The asynchronous one can be achieved using publisher/subscriber behaviour on shared topics. One of the nodes provided by ROS is *AMCL*, which provides an implementation of the *Adaptive Monte Carlo Localization* (a *particle filter* that estimates the current position and orientation of the robot inside a known map).

Another standard library is *tf2*, which allows the definition of a tree of reference systems (called *frames*), providing the ability to transform and combine data coming from different nodes. In Figure 3.1 a simplified representation (showing only the most relevant elements) of the system under study is shown, using the following conventions:

- a box represents a *topic*;
- a circle represents a *node*;
- an arrow going from a topic to a node indicates that this node is registered as a *subscriber* to that topic;
- an arrow going from a node to a topic indicates that this node is registered as a *publisher* on that topic.

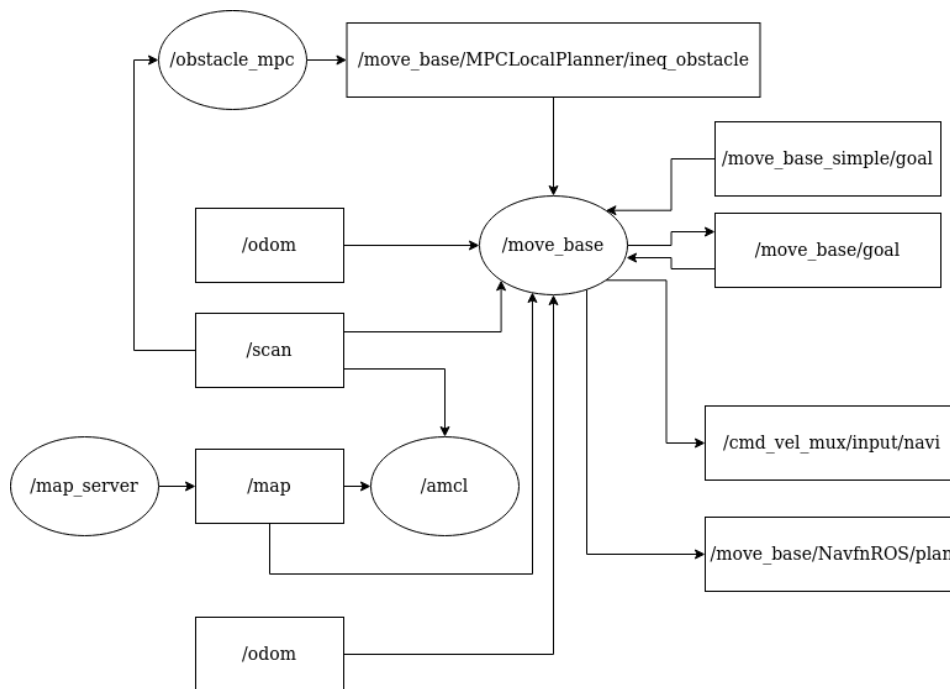


Figure 3.1: ROS system simplified architecture

To be specific, the *move_base* node, extensively described in [15], provides both global and local planning.

Global Planning is the procedure of finding a collision-free trajectory inside a known map.

In this work, the default package provided by ROS has been used, which

3.1 System Architecture

reads the desired goal on */move_base_simple/goal* topic and relies on an implementation of the A^* algorithm, introduced in [16], publishing the result on */move_base/NavfnROS/plan* topic.

This algorithm can be seen as an extension of Dijkstra's one, presented for the first time in [17], with the usage of a heuristic to guide its search, although it is more memory consuming ($\mathbf{O}(b^d)$ space complexity, where d is the depth of the solution and b the average number of successors per state, depending on the heuristic adopted) compared to other algorithms. On the other side, the Local Planner aims to follow the trajectory obtained from the Global Planner (i.e., issuing control commands to the robot), exploiting sensors data to detect obstacles and avoiding them while keeping safety and comfort constraints.

The Local Planner runs every τ_s seconds, builds the optimization problem defined in Eq. 3.1 and solves it interacting with the IBM iLOG CPLEX optimizer.

The *obstacle_mpc* node reads data from laser scanners (published on topic */scan*) and performs on them the procedures of *segmentation* (i.e., separate detected points in segments based on a maximum threshold) and *convexification* (i.e., detect concavities between segments and split in convex sets of lines) and then publishes data related to the constraints to be imposed to avoid obstacles, which will be shown later, on */move_base/MPCLocalPlanner/ineq_obstacle* topic.

Finally, *map_server* node provides map data on */map* topic and allows to interact with the map, allowing the *amcl* node, as described before, to estimate the current position.

In order to simulate an environment with the presence of moving obstacles, two libraries have been integrated.

The first one is *Pedsim*, which allows the definition of groups of pedestrians that move inside a known map, based on the force interaction model introduced in [18].

The second one is *find_moving_objects*, which reads data from */scan* topic and estimates the presence of moving objects using an Exponentially Weighted Moving Average, publishing the results on */moving_objects* topic. A more detailed description of the velocity detection algorithm can be found in [19].

3.2 Problem Definition

Model Predictive Control algorithms rely on a dynamic model of the system and a control problem expressed as an optimization one over a finite time horizon $[k, k + N]$.

The optimization problem is defined by a cost function J and a set of constraints, defined as functions of the vector of optimization variables v . In this work, a quadratically-constrained convex optimization problem is considered, with convex linear/quadratic equality/inequality constraints and a quadratic convex cost function:

$$\begin{aligned}
 \min_v \quad & J(v) = \frac{1}{2}v^T H v + f v \\
 \text{subject to} \quad & A_{ineq} v \leq b_{ineq} \\
 & A_{eq} v = b_{eq} \\
 & \mathcal{L} v + v^T \mathcal{Q} v \leq \mathcal{R} \\
 & lb \leq v \leq ub
 \end{aligned} \tag{3.1}$$

where the matrix H (Hessian) and the row vector f (gradient) represent the objective (cost) function $J(v)$ to be minimized.

While $A_{ineq}, b_{ineq}, A_{eq}, b_{eq}, l, Q, r, lb$ and ub are the matrices and vectors expressing linear inequality constraints (A_{ineq}, b_{ineq}), linear equality constraints (A_{eq}, b_{eq}), quadratic constraints ($\mathcal{L}, \mathcal{Q}, \mathcal{R}$), lower and upper bounds (lb, ub).

Another characteristic of MPC algorithms is the Receding Horizon principle. This consists in evaluating (at each time instant) the system in the future time window $[k, k + N]$ solving the optimization problem for the future control sequence $[u(k), \dots, u(k + N - 1)]$, and taking only the first element $u(k)$. In that way, a time-invariant feedback control strategy is obtained.

Assuming that the state $x(k) \in \mathbb{R}^n$ is measurable, and that the control variable is $u(k) \in \mathbb{R}^m$, a generic linear discrete time system can be considered:

$$\begin{cases} x(k+1) & = Ax(k) + Bu(k) \\ y(k) & = Cx(k) \end{cases} \tag{3.2}$$

And a cost function J defined over the selected prediction horizon N :

3.2 Problem Definition

$$J = \sum_{i=0}^{N-1} l(x(k+i), u(k+i)) + V^f(x(k+N)) \quad (3.3)$$

where $l(x, u)$ is a positive definite function called stage cost, and $V^f(x(k+N))$ is the terminal cost. In this work, a quadratic cost function is considered:

$$J(x(k), u(k)) = \sum_{i=0}^{N-1} (\|x(k+i)\|_Q^2 + \|u(k+i)\|_R^2) + \|x(k+N)\|_S^2 \quad (3.4)$$

where $Q = Q^T \geq 0$, $R = R^T \geq 0$ and $S = S^T \geq 0$ are weight matrices of suitable dimension, and the expression:

$$\|x\|_Z^2 = x^T Z x \quad (3.5)$$

Constraints on state and control variable must be in the form:

$$x(k) \in \mathbb{X} \subset \mathbb{R}^n \quad (3.6)$$

$$u(k) \in \mathbb{U} \subset \mathbb{R}^m \quad (3.7)$$

where \mathbb{X} and \mathbb{U} are state and control variables trust region. These constraints must be expressed in a way that is compatible with the optimization problem (i.e., convex equality or inequality constraints).

As described in [20], the idea of stability in an MPC controller is related to the Recursive Feasibility principle. Given a state variable $x(k)$ such that:

$$x(k) \in \mathbb{X} \subset \mathbb{R}^n \quad (3.8)$$

the terminal set can be defined as

$$\mathbb{X}^f \subset \mathbb{X} \quad (3.9)$$

In particular, to ensure the solvability of the optimization problem, it is required that the state at the end of the prediction horizon, belongs to the terminal set:

$$x(k + N) \in \mathbb{X}^f \quad (3.10)$$

The terminal set \mathbb{X}^f is designed with reference to the generic discrete-time system 3.2 and the auxiliary time-varying optimal control law:

$$\begin{aligned} u^o(k + i) &= -K(i)x(k + 1) \\ \forall i &\in 0, \dots, N - 1 \end{aligned} \quad (3.11)$$

Values of K must be chosen to ensure that the eigenvalues of the matrix $A - BK$ guarantee the stability of the closed-loop system:

$$x(k + 1) = (A - BK)x(k) \quad (3.12)$$

Then, recalling the penalty matrices Q and K introduced in the cost function, and taking into account the Discrete Riccati Equation:

$$(A - BK)^T S (A - BK) - S = -(Q + K^T R K) \quad (3.13)$$

the positive definite weight matrix S , associated with the terminal cost in Eq. 3.3, can be found.

The terminal set \mathbb{X}^f is said to be positive invariant with respect to the closed-loop system in Eq. 3.13 if:

$$x(\bar{k}) \in \mathbb{X}^f \Rightarrow x(k) \in \mathbb{X}^f, \quad \forall k \geq \bar{k} \quad (3.14)$$

Finally, it is required that:

$$u(k) = Kx(k) \subseteq \mathbb{U}, \quad \forall x(k) \in \mathbb{X}^f \quad (3.15)$$

In that way, starting from an initial state $x(\bar{k}) \in \mathbb{X}^f$ and applying, eventually, the auxiliary control law, the state will belong to the terminal set \mathbb{X}^f , guaranteeing control requirements. Finally, Model Predictive Control is suitable for the implementation of *slack variables*, these are additional optimization variables, which are bounded to vary between 0 and 1:

$$0 \leq u_{sl}^j(k) \leq 1, \quad \forall j \quad (3.16)$$

These variables can be used to relax constraints, in case problem optimality is compromised, as it will be described later.

3.3 Autonomous wheelchair model

Since this work is based on [8] and [13], the same wheelchair will be considered as the reference model.

This wheelchair is a Degonda Twist t4 2x2 (produced by Degonda Rehab SA, see Figure 3.2), designed for indoor and outdoor mobility, with two rear driving wheels, whose motors have a maximum power of $0.35[kW]$, and three caster stabilizing wheels. Additionally, it has been equipped with two SICK TiM561 time-of-flight laser sensors, positioned at two opposite corners, in order to guarantee a complete 360° vision of the surrounding environment, within a radius of $10[m]$, as shown in Figure 3.3.



Figure 3.2: Degonda Twist t4 2x2

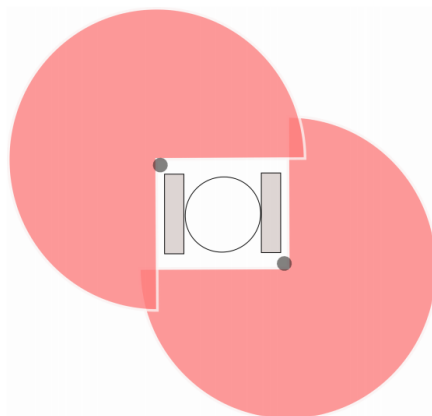


Figure 3.3: Laser sensors 360° coverage

From a kinematic point of view, this wheelchair can be represented as a differential-drive vehicle (see Figure 3.4), which has two separately controlled fixed wheels with a common axis of rotation, and one or more passive caster wheels to keep the robot statically balanced.

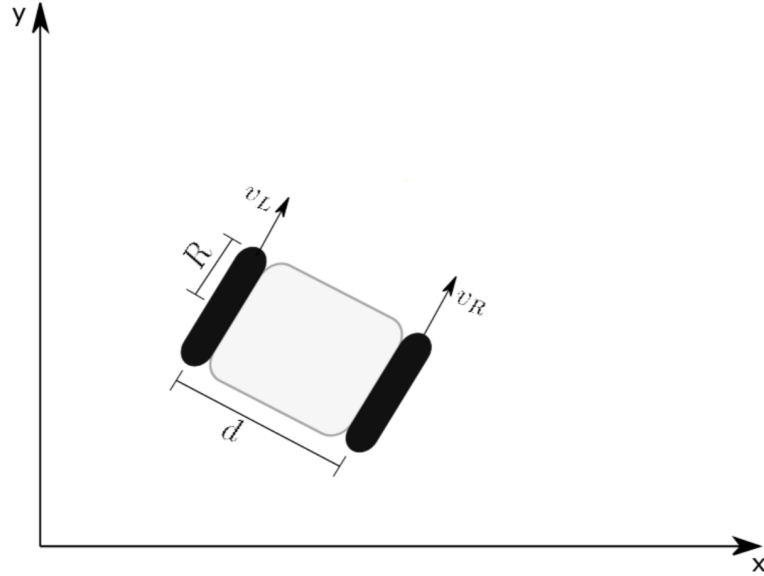


Figure 3.4: Differential drive model, R is wheel radius and d the distance between wheels

The two fixed wheels are actuated by two independent motors that provide to wheels the rotational velocities ω_R and ω_L , resulting in the linear velocities:

$$\begin{cases} v_R = \omega_R R \\ v_L = \omega_L R \end{cases} \quad (3.17)$$

This vehicle control variables are linear velocity v and angular velocity ω , which can be related to wheel velocities by the following relations:

$$\begin{cases} v = R \frac{\omega_R + \omega_L}{2} \\ \omega = R \frac{\omega_R - \omega_L}{2} \end{cases} \quad (3.18)$$

As a consequence, the differential drive model can be represented in an

3.3 Autonomous wheelchair model

equivalent unicycle model (see Figure 3.5):

$$\begin{cases} \dot{x}(t) = v(t)\cos(\theta(t)) \\ \dot{y}(t) = v(t)\sin(\theta(t)) \\ \dot{\theta}(t) = \omega(t) \end{cases} \quad (3.19)$$

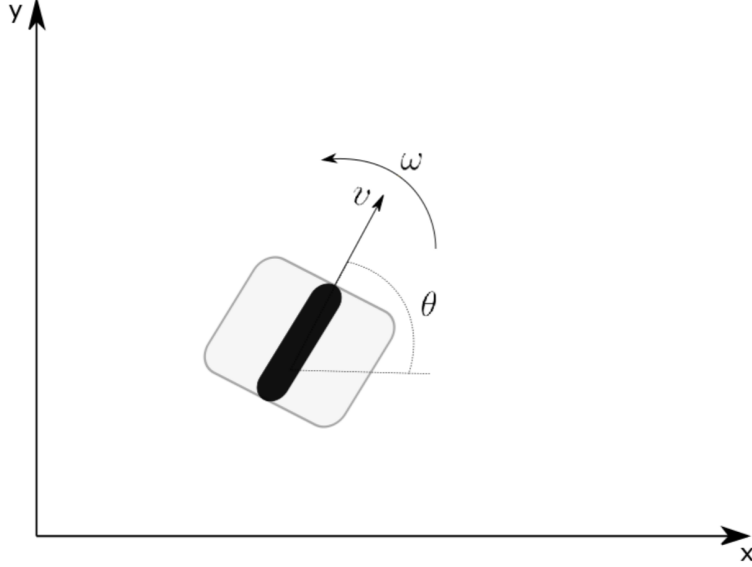


Figure 3.5: Unicycle

where $x(t)$ and $y(t)$ represent robot center position and $\theta(t)$ its orientation in the global reference system, while the linear velocity $v(t)$ and the angular velocity $\omega(t)$ are input variables.

As it can be seen from 3.19, there is a nonlinear relationship between state and control variables, that is not compatible with the development of a linear controller.

As shown in [21], this system can be transformed to a particle, thanks to an inner feedback-linearization loop, showing a linear dynamics under a suitable change of variables. This can be achieved defining a point P placed at distance ε from the unicycle wheel axle center, in direction of the linear velocity (see Figure 3.6), having the following coordinates in the global reference system:

$$\begin{cases} x_P(t) = x(t) + \varepsilon\cos(\theta(t)) \\ y_P(t) = y(t) + \varepsilon\sin(\theta(t)) \end{cases} \quad (3.20)$$

By deriving with respect to time:

$$\begin{cases} \dot{x}_P(t) = \dot{x}(t) + \varepsilon \sin(\theta(t)) \dot{\theta}(t) \\ \dot{y}_P(t) = \dot{y}(t) + \varepsilon \sin(\theta(t)) \dot{\theta}(t) \end{cases} \quad (3.21)$$

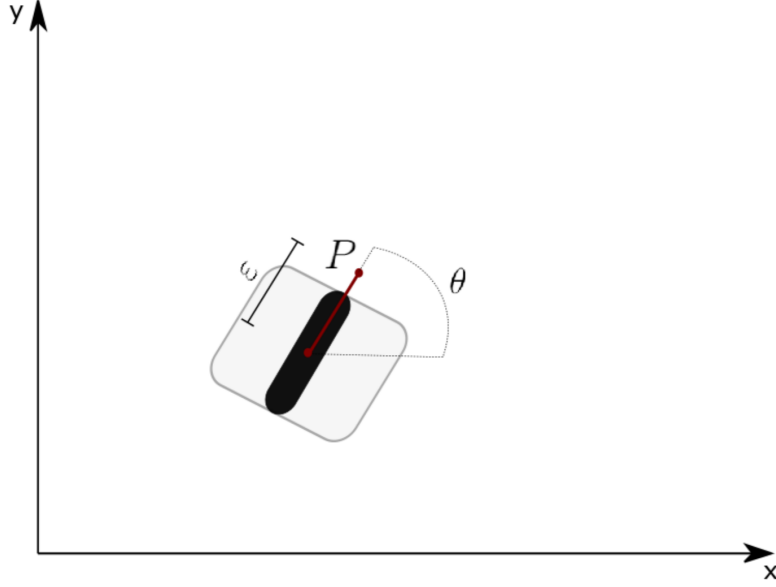


Figure 3.6: Feedback Linearization model

Then, by substituting the unicycle model 3.19, the relation 3.21 can be rewritten in matrix form as:

$$\begin{bmatrix} \dot{x}_P(t) \\ \dot{y}_P(t) \end{bmatrix} = \begin{bmatrix} v_{P_x}(t) \\ v_{P_y}(t) \end{bmatrix} = \begin{bmatrix} \cos(\theta(t)) & -\varepsilon \sin(\theta(t)) \\ \sin(\theta(t)) & \varepsilon \cos(\theta(t)) \end{bmatrix} \begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} \quad (3.22)$$

From which it is possible to retrieve the Feedback Linearization transformation matrix:

$$T(\theta, \varepsilon) = \begin{bmatrix} \cos(\theta(t)) & -\varepsilon \sin(\theta(t)) \\ \sin(\theta(t)) & \varepsilon \cos(\theta(t)) \end{bmatrix} \quad (3.23)$$

Matrix $T(\theta, \varepsilon)$ is invertible $\forall \theta$ and $\varepsilon \neq 0$, thus it is possible to relate the variables of the linearized system ($v_{P_x}(t)$ and $v_{P_y}(t)$) with the real ones ($v(t)$ and $\omega(t)$), obtaining a linear decoupled system, characterized by two integrators:

$$\begin{cases} \dot{x}_P(t) = v_{P_x}(t) \\ \dot{y}_P(t) = v_{P_y}(t) \end{cases} \quad (3.24)$$

3.4 Cost function

This model is continuous-time, and it needs to be discretized in order to be compatible with a discrete-time control approach. In this work, the Forward Euler method is used, which consists in imposing

$$s = \frac{z - 1}{\tau} \quad (3.25)$$

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

$$\begin{cases} x_P(k+1) = x_P(k) + \tau v_{P_x}(k) \\ y_P(k+1) = y_P(k) + \tau v_{P_y}(k) \end{cases} \quad (3.26)$$

which can be rewritten in matrix form as:

$$\xi(k+1) = A\xi(k) + Bu(k) \quad (3.27)$$

where

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, B = \begin{bmatrix} \tau & 0 \\ 0 & \tau \end{bmatrix}, \xi(k) = \begin{bmatrix} x_P(k) \\ y_P(k) \end{bmatrix}, u(k) = \begin{bmatrix} v_{P_x}(k) \\ v_{P_y}(k) \end{bmatrix} \quad (3.28)$$

It is worth to notice that the linearized model in Eq. 3.21 has only two (position) state variables, while the original one (Eq. 3.19) had three variables. The change of coordinates has brought a loss of observability, and thus the robot heading is no more observable from the output.

3.4 Cost function

As introduced before, Model Predictive Control is based on the definition of an optimization problem, as the one in Eq. 3.1. To this aim, a cost function is designed to fulfil system variables requirements, to guarantee the optimal convergence of the state variables of interest to the desired reference value, and to avoid excessive fluctuation in the control variables. In this work, a quadratic cost function is considered:

$$J(k) = \sum_{i=0}^{N-1} (\|\xi(k+i) - \xi_{ref}(k+i)\|_Q^2 + \|u(k+i)\|_R^2) + \|\xi(k+N) - \xi_{ref}(k+N)\|_S^2 \quad (3.29)$$

where $\xi_{ref}(k+i)$ is the position reference computed by the Global Planner at the considered time instant

$$\xi_{ref}(k+i) = \begin{bmatrix} x_{ref}(k+i) \\ y_{ref}(k+i) \end{bmatrix} \quad (3.30)$$

and Q , R and S are weight matrices that must be tuned to ensure the fulfillment of the requirements on state, control variables and stability. In order to allow the optimization solver to find a solution, the cost function is formalized as an open loop solution, derived by computing the prediction of the future states based on the value of the current state $\xi(k)$. In fact, recalling Lagrange Equation:

$$\xi(k+i) = A^i \xi(k) + \sum_{j=0}^{i-1} A^{i-j-1} B u(k+j) \quad , i > 0 \quad (3.31)$$

and letting:

$$\begin{aligned} \mathcal{A} &= \begin{bmatrix} I \\ A \\ A^2 \\ \vdots \\ A^N \end{bmatrix}_{(2(N+1),2)} & \mathcal{B} &= \begin{bmatrix} 0 & 0 & \dots & 0 \\ B & 0 & \dots & 0 \\ AB & B & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{N-1}B & A^{N-2}B & \dots & B \end{bmatrix}_{(2(N+1),2N)} \\ \Xi(k) &= \begin{bmatrix} \xi(k) \\ \vdots \\ \xi(k+N) \end{bmatrix}_{(2(N+1),1)} & \mathcal{U}(k) &= \begin{bmatrix} u(k) \\ \vdots \\ u(k+N-1) \end{bmatrix}_{(2N,1)} \end{aligned} \quad (3.32)$$

it follows that:

$$\Xi(k) = \mathcal{A}\xi(k) + \mathcal{B}\mathcal{U}(k) \quad (3.33)$$

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

$$\begin{aligned} \tilde{J}(\Xi(k), \mathcal{U}(k)) &= \|\Xi(k) - \Xi_{ref}\|_Q^2 + \|\mathcal{U}(k)\|_R^2 = \\ &= (\Xi(k) - \Xi_{ref})^T \mathcal{Q} (\Xi(k) - \Xi_{ref}) + \mathcal{U}^T(k) \mathcal{R} \mathcal{U}(k) \end{aligned} \quad (3.34)$$

where

$$\mathcal{Q} = \begin{bmatrix} Q & & & \\ & \ddots & & \\ & & Q & \\ & & & S \end{bmatrix}_{(2(N+1),2(N+1))} \quad \mathcal{R} = \begin{bmatrix} R & & \\ & \ddots & \\ & & R \end{bmatrix}_{(2N,2N)} \quad (3.35)$$

Finally, by recalling Eq. 3.33, Eq. 3.34 can be rewritten as:

$$\begin{aligned} \tilde{J}(k) &= (\mathcal{A}\xi(k) + \mathcal{B}\mathcal{U}(k) - \Xi_{ref})^T \mathcal{Q} (\mathcal{A}\xi(k) + \mathcal{B}\mathcal{U}(k) - \Xi_{ref}) + \\ &\quad + \mathcal{U}^T(k) \mathcal{R} \mathcal{U}(k) \end{aligned} \quad (3.36)$$

and a cost function suitable for the MPC formulation is obtained:

$$\tilde{J} = \mathcal{U}^T(k) H \mathcal{U}(k) + 2f^T \mathcal{U}(k) + cost \quad (3.37)$$

3.5 Velocity constraints

where

$$H = \mathcal{B}^T \mathcal{Q} \mathcal{B} + \mathcal{R} \quad , \quad f = (\mathcal{A}\xi(k) - \Xi_{ref})^T \mathcal{Q} \mathcal{B} \quad (3.38)$$

Additionally, considering the presence of a vector of n_{sl} slack variables u_{sl_p} , the cost function can be extended as:

$$\bar{J}(k) = J(k) + \|u_{sl_p}(k)\|_{S_p}^2 \quad (3.39)$$

where

$$S_p = \begin{bmatrix} s_p & & \\ & \ddots & \\ & & s_p \end{bmatrix}_{(n_{sl}, n_{sl})} \quad (3.40)$$

S_p is a weight matrix, whose coefficient will be chosen very high with respect to the values of q and r , in that way, the optimizer will select values different from zero for the slack variables, only if problem optimality is compromised.

Finally, it is possible to define the new cost function as:

$$\bar{J}(k) = \frac{1}{2} \bar{\mathcal{U}}(k)^T \bar{H} \bar{\mathcal{U}}(k) + 2 \bar{f}^T \bar{\mathcal{U}}(k) + cost \quad (3.41)$$

where

$$\bar{H} = \begin{bmatrix} H & 0_{(2N, n_{sl})} \\ 0_{(n_{sl}, 2N)} & S_p \end{bmatrix}_{(2N+n_{sl}, 2N+n_{sl})} \quad (3.42)$$

$$\bar{f}^T = \begin{bmatrix} f^T & 0_{(1, n_{sl})} \end{bmatrix}_{(1, 2N+n_{sl})} \quad (3.43)$$

and

$$\bar{\mathcal{U}}(k) = \begin{bmatrix} \mathcal{U}(k) \\ u_{sl_p}(k) \end{bmatrix}_{(2N+n_{sl}, 1)} \quad (3.44)$$

3.5 Velocity constraints

3.5.1 Quadratic velocity constraints

As explained in [7], it is possible to define a set of quadratic constraints that limit control variables v_{P_x} and v_{P_y} in order to avoid having the robot exceeding a maximum linear or angular velocity. Hence, $\forall i \in \{0, \dots, N-1\}$ it is required that:

$$0 \leq \sqrt{v_{P_x}^2(k+i) + v_{P_y}^2(k+i)} \leq v_{max} \quad (3.45)$$

and, recalling Eq. 3.1, quadratic constraints in control variables must be expressed as a set of inequalities having the following form:

$$\mathcal{L}_i u(k+i) + u(k+i)^T \mathcal{Q}_i u(k+i) \leq \mathcal{R}_i \quad (3.46)$$

with

$$u(k+i) = \begin{bmatrix} v_{P_x}(k+i) \\ v_{P_y}(k+i) \end{bmatrix} \quad (3.47)$$

where \mathcal{L}_i represents the linear part of the i -th constraint, \mathcal{Q}_i the quadratic part, and \mathcal{R}_i the scalar one.

This velocity constraints can be rewritten as N quadratic constraints, $\forall i \in \{0, \dots, N-1\}$:

$$\begin{bmatrix} v_{P_x}(k+i) & v_{P_y}(k+i) \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_{P_x}(k+i) \\ v_{P_y}(k+i) \end{bmatrix} \leq v_{max}^2 \quad (3.48)$$

with

$$\mathcal{L}_i = \begin{bmatrix} 0 & 0 \end{bmatrix}, \mathcal{Q}_i = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \mathcal{R}_i = \begin{bmatrix} v_{max}^2 \end{bmatrix} \quad (3.49)$$

3.5.2 Linear velocity constraints

Another formulation of velocity constraints presented in [7] exploits the Eq. [21], rewriting it as:

$$\begin{cases} v = v_{P_x} \cos(\theta) + v_{P_y} \sin(\theta) \\ \omega = \frac{1}{\varepsilon} (v_{P_y} \cos(\theta) - v_{P_x} \sin(\theta)) \end{cases} \quad (3.50)$$

Recalling the Differential Drive model, the following relations can be obtained by substitution:

$$\begin{aligned} \omega_R &= \frac{1}{2R} (2\cos(\theta) - \frac{d}{\varepsilon} \sin(\theta)) v_{P_x} + \frac{1}{2R} (2\sin(\theta)) + \frac{d}{\varepsilon} \cos(\theta) v_{P_y} \\ \omega_L &= \frac{1}{2R} (2\cos(\theta) + \frac{d}{\varepsilon} \sin(\theta)) v_{P_x} + \frac{1}{2R} (2\sin(\theta)) - \frac{d}{\varepsilon} \cos(\theta) v_{P_y} \end{aligned} \quad (3.51)$$

These can be used to impose constraints in the form of:

$$\begin{aligned} \bar{\omega}_m &\leq \omega_R \leq \bar{\omega}_M \\ \bar{\omega}_m &\leq \omega_L \leq \bar{\omega}_M \end{aligned} \quad (3.52)$$

In fact, assuming the orientation θ to be known, they can be written as:

$$\frac{1}{2R} \begin{bmatrix} 2\cos(\theta) - \frac{d}{\varepsilon} \sin(\theta) & 2\sin(\theta) + \frac{d}{\varepsilon} \cos(\theta) \\ 2\cos(\theta) + \frac{d}{\varepsilon} \sin(\theta) & 2\sin(\theta) - \frac{d}{\varepsilon} \cos(\theta) \end{bmatrix} \begin{bmatrix} v_{P_x} \\ v_{P_y} \end{bmatrix} \leq \begin{bmatrix} \bar{\omega}_M \\ \bar{\omega}_M \end{bmatrix} \quad (3.53)$$

3.6 Acceleration constraints

$$-\frac{1}{2R} \begin{bmatrix} 2\cos(\theta) - \frac{d}{\varepsilon}\sin(\theta) & 2\sin(\theta) + \frac{d}{\varepsilon}\cos(\theta) \\ 2\cos(\theta) + \frac{d}{\varepsilon}\sin(\theta) & 2\sin(\theta) - \frac{d}{\varepsilon}\cos(\theta) \end{bmatrix} \begin{bmatrix} v_{P_x} \\ v_{P_y} \end{bmatrix} \leq \begin{bmatrix} -\bar{\omega}_m \\ -\bar{\omega}_m \end{bmatrix} \quad (3.54)$$

Then, defining:

$$\bar{A}_i = \frac{1}{2R} \begin{bmatrix} 2\cos(\theta) - \frac{d}{\varepsilon}\sin(\theta) & 2\sin(\theta) + \frac{d}{\varepsilon}\cos(\theta) \\ 2\cos(\theta) + \frac{d}{\varepsilon}\sin(\theta) & 2\sin(\theta) - \frac{d}{\varepsilon}\cos(\theta) \end{bmatrix} \quad (3.55)$$

and, in order to be compatible with the optimization problem formulation, Eq. 3.1, they must be expressed as:

$$\bar{\mathcal{A}}_{vel}\mathcal{U}(k) \leq \bar{b}_{vel} \quad (3.56)$$

Where

$$\mathcal{U}(k) = \begin{bmatrix} u(k) \\ u(k+1) \\ \vdots \\ u(k+N) \end{bmatrix}_{(2(N+1),1)} \quad (3.57)$$

and

$$\bar{A}_{vel} = \begin{bmatrix} \bar{A}_1 & 0 & \dots & 0 \\ 0 & \bar{A}_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \bar{A}_N \\ -\bar{A}_1 & 0 & \dots & 0 \\ 0 & -\bar{A}_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & -\bar{A}_N \end{bmatrix}_{(4N,2N)} \quad \bar{b}_{vel} = \begin{bmatrix} \bar{\omega}_M \\ \vdots \\ \vdots \\ \bar{\omega}_M \\ -\bar{\omega}_m \\ \vdots \\ \vdots \\ -\bar{\omega}_m \end{bmatrix}_{(4N,1)} \quad (3.58)$$

The values of $\theta(i)$ can be estimated considering the values of the control sequence at the previous time instant $\mathcal{U}(k-1)$.

3.6 Acceleration constraints

To guarantee that Model Predictive Control finds a solution compatible with system dynamics (in terms of maximum allowed acceleration) and to satisfy comfort and trajectory smoothness requirements, the maximum allowed velocity variation between two consecutive time instants must be

3.7 Position constraints

Then, recalling the form of linear inequality constraints in Eq. 3.1, the condition to be imposed must be expressed as:

$$A_{\Delta v}\mathcal{U}(k) \leq b_{\Delta v} \quad (3.65)$$

Which can be obtained by rearranging terms in Eq. 3.64 and finding that:

$$A_{\Delta v} = \left[A_{var}V \right]_{(4N,2N)} \quad b_{\Delta v} = \left[\Delta V + A_{var}v_0 \right]_{(4N,1)} \quad (3.66)$$

3.7 Position constraints

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

$$x(k) \in \mathbb{X} \quad (3.67)$$

defines a trust region representing the space of the environment in which the robot is allowed to move.

In particular, the definition of hyperplanes that separate the convex region containing the wheelchair from the detected obstacles can be used to avoid collisions.

Considering vehicle configuration space, the hyperplanes that delimit the trust region has the form:

$$h_x x + h_y y \leq l \quad (3.68)$$

where h_x and h_y are the coefficients related to variables x and y , respectively, representing the slope of a 1-dimensional line, and l is the distance from the origin.

In particular, to ensure the solvability of the optimization problem, the vehicle must be contained in the half plane defined by the constraint, thus the following condition must hold:

$$h_x \hat{x}_c + h_y \hat{y}_c \leq l \quad (3.69)$$

where \hat{x}_c and \hat{y}_c are the coordinates of the estimated position of the vehicle. If that condition does not hold, the signs of the inequality must be inverted (i.e., selecting the other half plane delimited by the line), obtaining:

$$(-h_x)\hat{x}_c + (-h_y)\hat{y}_c \leq (-l) \quad (3.70)$$

Then, recalling that constraints can be imposed on each time instant within the selected prediction horizon N , for each obstacle, the constraint has the form:

$$h_x^{(i)(j)}x(k+i) + h_y^{(i)(j)}y(k+i) \leq l^{(i)(j)} \quad \forall i \in \{0, \dots, N\}, \forall j \in \{0, \dots, n_{obs}\} \quad (3.71)$$

where $h_x^{(i)(j)}$, $h_y^{(i)(j)}$ and $l^{(i)(j)}$ represent the coefficients of the j -th constraint at instant $k+i$.

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.

By recalling Lagrange Equation, constraints on the state variables along the entire prediction horizon can be expressed as:

$$\underbrace{\begin{bmatrix} h_{k|k-1}^{(j)} \\ \vdots \\ h_{k+N|k-1}^{(j)} \end{bmatrix}}_{H_{obs}^{(j)}} \underbrace{\begin{bmatrix} \xi(k) \\ \vdots \\ \xi(k+N) \end{bmatrix}}_{\Xi(k)} \leq \underbrace{\begin{bmatrix} l_{k|k-1}^{(j)} \\ \vdots \\ l_{k+N|k-1}^{(j)} \end{bmatrix}}_{L_{obs}^{(j)}} \quad (3.72)$$

where $[H_{obs}^{(j)}]_{(N+1, 2(N+1))}$ and $[L_{obs}^{(j)}]_{(N+1, 1)}$ represent the matrices of the coefficients related to the j -th state constraint. These coefficients have the subscript $k+i|k-1$ since they represent their value at instant $k+i$ based on the state prediction computed at time $k-1$, and, in particular:

$$h_{k+i|k-1}^{(j)} = \begin{bmatrix} h_{x_{k+i|k-1}}^{(j)} & h_{y_{k+i|k-1}}^{(j)} \end{bmatrix} \quad (3.73)$$

In addition, slack variables could also be used to take into account a safety distance, which eventually could be violated by the optimizer, if needed:

$$h_x^{(i)(j)}x(k+i) + h_y^{(i)(j)}y(k+i) \leq l^{(i)(j)} - \Delta l^{(i)(j)}(1 - u_{sl_p}^{(j)}(k)) \quad (3.74)$$

$$\forall i \in \{0, \dots, N\}, \forall j \in \{0, \dots, n_{obs}\}$$

with

$$\Delta l^{(i)(j)} = \Delta l_{sl}^{(i)(j)} \sqrt{(h_x^{(i)(j)})^2 + (h_y^{(i)(j)})^2} \quad (3.75)$$

and $\Delta l_{sl}^{(i)(j)}$ the value of the safety distance to be imposed. In particular, defining an extraction matrix $E^{(j)}$:

$$E^{(j)} = \begin{bmatrix} 0 & \dots & 0 & \dots & -\Delta l^{(i)(j)} & 0 & \dots & 0 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \\ 0 & \dots & 0 & \dots & \underbrace{-\Delta l^{(i)(j)}}_{\text{j-th position}} & 0 & \dots & 0 \end{bmatrix}_{(N+1, n_{obs})} \quad (3.76)$$

3.7 Position constraints

where the only column having values different from zero is the j -th one, allowing to formulate:

$$u_{sl_p}^{(j)}(k) = E^{(j)}u_{sl_p}(k) \quad (3.77)$$

where $u_{sl_p}(k)$ is the vector of slack variables.

Then, the vector of constant terms of the constraints can be extended, taking into account the safety distance:

$$\bar{L}_{obs}^{(j)} = L_{obs}^{(j)} - \begin{bmatrix} \Delta l^{(j)} \\ \vdots \\ \Delta l^{(j)} \end{bmatrix}_{(N+1,1)} \quad (3.78)$$

And, recalling the enlarged set of control variables $\bar{\mathcal{U}}(k)$ defined in Eq. 3.44:

$$\bar{\mathcal{U}}(k) = \begin{bmatrix} \mathcal{U}(k) \\ u_{sl_p}(k) \end{bmatrix}_{(2N+n_{obs},1)} \quad (3.79)$$

The constraints can be rewritten in a suitable form for the Model Predictive Control as:

$$\bar{A}_{obs}^{(j)}\bar{\mathcal{U}}(k) \leq \bar{b}_{obs}^{(j)} \quad (3.80)$$

with

$$\bar{A}_{obs}^{(j)} = \begin{bmatrix} H_{obs}^{(j)}\mathcal{B} & E^{(j)} \end{bmatrix}_{(N+1,2N+n_{obs})} \quad \bar{b}_{obs}^{(j)} = \bar{L}_{obs}^{(j)} - H_{obs}^{(j)}\mathcal{A}\xi(k) \quad (3.81)$$

3.7.1 Socially-compliant constraints

The work done in [7] has shown the definition of state constraints allowing to impose a safety distance from obstacles, compliant with social requirements, like not violating personal space boundaries (see Figure 3.7). Before going into the description of the constraints, a mathematical formulation of the personal space function will be given.

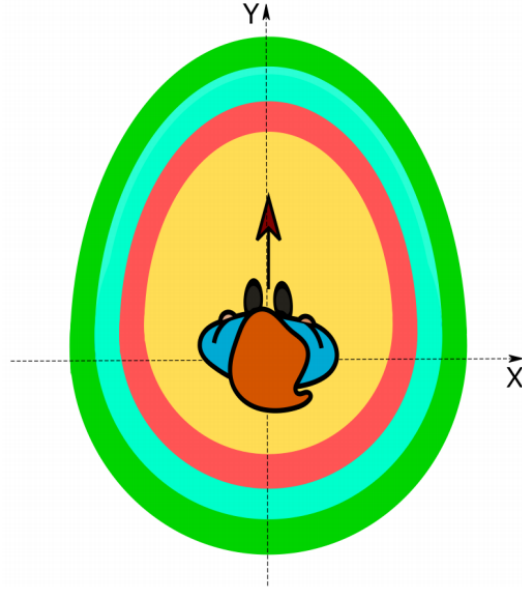


Figure 3.7: Personal Space function

Personal space function can be defined as a 2-D asymmetric Gaussian, as shown in [22]:

$$PSf(x, y) = e^{-\left(\frac{(x-x_0)^2}{2\sigma_x^2} + \frac{(y-y_0)^2}{2\sigma_y^2}\right)} \quad (3.82)$$

with varying variances σ_x and σ_y and mean (x_0, y_0) . In order to relate variances with the rotation of the function, the following coefficients has to be defined (see Figure 3.8):

- θ^{po} : rotation of the function minus $\frac{\pi}{2}$;
- σ_l : variance to the right side (with respect to θ^{po} direction);
- σ_h : variance along the rotation of function ($\theta^{po} + \frac{\pi}{2}$ direction);
- σ_s : variance to the left side ($\theta^{po} + \pi$ direction);
- σ_r : variance to the rear ($\theta^{po} - \frac{\pi}{2}$ direction).

3.7 Position constraints

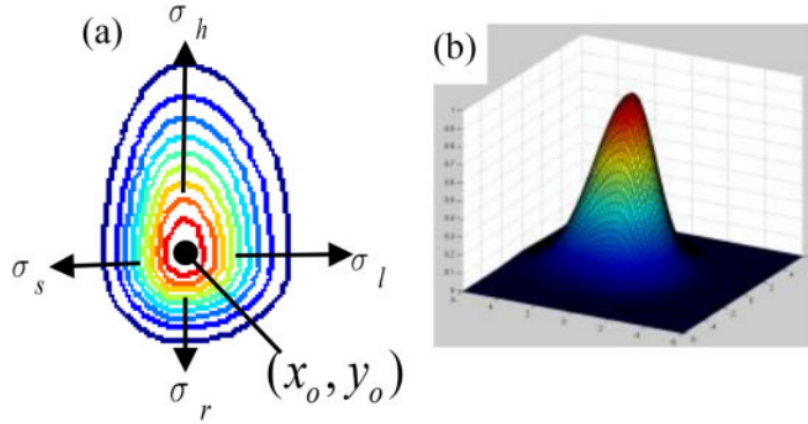


Figure 3.8: Contour (a) and surface maps (b) of the two-dimensional asymmetric Gaussian function

In particular, by defining:

$$\begin{aligned}
 d &= \sqrt{(x - x_0)^2 + (y - y_0)^2} \\
 \theta &= \text{atan2}((y - y_0), (x - x_0)) \\
 \alpha &= \theta - \theta^{p_0}, \quad \alpha \in (-\pi, \pi]
 \end{aligned} \tag{3.83}$$

the personal space function $PSf(x, y)$ can be rewritten as:

$$PSf(d, \alpha) = e^{-\left(\frac{(d \cos(\alpha))^2}{2\sigma_x^2} + \frac{(d \sin(\alpha))^2}{2\sigma_y^2}\right)} \tag{3.84}$$

where the values of σ_x and σ_y vary according to α in the different quadrants:

$$\begin{cases}
 \sigma_x = \sigma_l, & \sigma_y = \sigma_h, & 0 \leq \alpha \leq \frac{\pi}{2} \\
 \sigma_x = \sigma_s, & \sigma_y = \sigma_h, & \frac{\pi}{2} < \alpha \leq \pi \\
 \sigma_x = \sigma_s, & \sigma_y = \sigma_r, & -\pi < \alpha < -\frac{\pi}{2} \\
 \sigma_x = \sigma_l, & \sigma_y = \sigma_r, & -\frac{\pi}{2} \leq \alpha < 0
 \end{cases} \tag{3.85}$$

According to the work done in [22], the following experimental coefficients has been defined (see Figure ??):

- l_r : length of intimate zone box, $l_r = 0.5[\text{m}]$;
- d_h : distance between frontal limit and intimate zone box, where:

$$d_h = 1.2[\text{m}] + v \times 1[\text{s}] \tag{3.86}$$

with v the movement speed in $[\frac{\text{m}}{\text{s}}]$;

- d_r : distance between rear limit and intimate zone box, $d_r = 0.8[\text{m}]$;

- d_l : distance between right limit and intimate zone box, $d_l = 0.8[\text{m}]$;
- d_s : distance between left limit and intimate zone box, $d_s = 0.5[\text{m}]$ (notice that $d_s < d_l$ since the social convention in China is to pass pedestrians on the left).

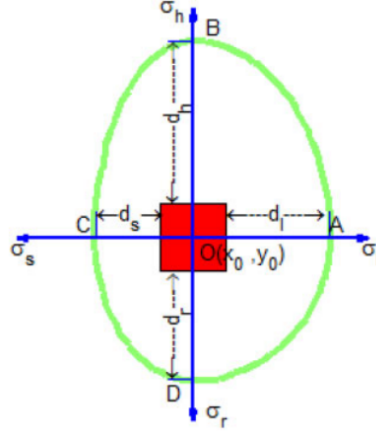


Figure 3.9: Human social interaction space

Finally, the four variances σ_l , σ_h , σ_s and σ_r can be determined from the corresponding experimental coefficients d_l , d_h , d_s and d_r as follows:

$$\sigma = \frac{1}{2} \sqrt{\frac{(d + 0.5l_r)^2}{\ln(10)}} \quad (3.87)$$

Due to this definition of σ coefficients, the points on the boundaries of the personal space correspond to:

$$e^{-(\ln(10)+\ln(10))} = 0.01$$

By imposing this value in Eq. 3.82, and applying $\ln(\cdot)$ to both members, the equation of the ellipsis describing personal space boundary is found:

$$\frac{(x - x_0)^2}{2\sigma_x^2} + \frac{(y - y_0)^2}{2\sigma_y^2} = -\ln(0.01) \quad (3.88)$$

The constraint line to be imposed can be computed as the blue line shown in Figure 3.10. This one is evaluated as the line tangent to obstacle's Virtual Box (i.e., the circumference that inscribes it, \hat{B} , enlarged by robot radius) passing through the intersection between point the relative velocity $v_{A,B}$ and \hat{B} .

3.7 Position constraints

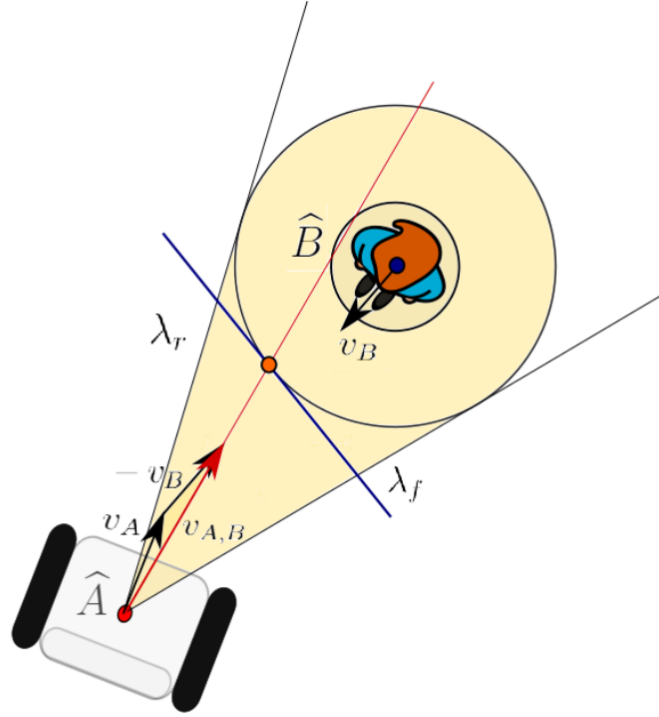


Figure 3.10: Position constraint determination (the blue line tangent to \hat{B})

In particular, the final constraint to be imposed, has the following form:

$$h_x + h_y \leq l - \Delta l \quad (3.89)$$

where $\Delta l = \Delta l_{sl} \sqrt{h_x^2 + h_y^2}$, and Δl_{sl} is a safety distance taking into account personal space function, that can be evaluated with the following algorithm:

Algorithm 1: Δl_{sl} determination

- 1 $P_1 \leftarrow \text{GetIntersectionPoint}(v_r, VB)$;
 - 2 $l \leftarrow \text{GetLineFrom2Points}(P_1, P_p)$;
 - 3 $P_2 \leftarrow \text{GetIntersectionPoint}(l, \text{reduced}VB)$;
 - 4 $P_3 \leftarrow \text{GetIntersectionPoint}(l, PSf)$;
 - 5 $\Delta l_{sl} \leftarrow \text{EuclideanDistance}(P_2, P_3)$;
 - 6 **return** Δl_{sl}
-

where the following auxiliary functions have been used:

- *GetIntersectionPoint*: function that given two functions in \mathbb{R}^2 finds the intersection points, if there exists multiple intersection points, the closer one to the robot is taken;

- *GetLineFrom2Points*: function that given two points in \mathbb{R}^2 computes the coefficients of the line passing through them;
- *EuclideanDistance*: function that given two points in \mathbb{R}^2 computes the euclidean distance between them.

The resulting safety distance Δl_{sl} can be seen in Figure 3.11 as the distance between the two green points.

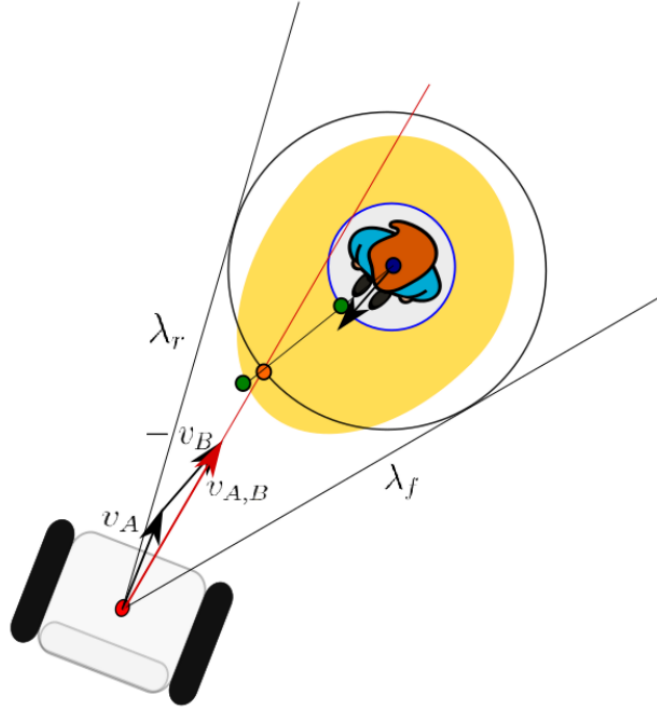


Figure 3.11: Δl_{sl} is determined as the euclidean distance between the points highlighted by the two green circles, representing the intersections between the pedestrian sensor virtual box (blue circle) and the Pedestrian personal space function (in yellow), respectively, with the black line defined joining 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.

3.7.2 Maximally violated line constraints

With reference to the work done in [8] and [13], after the procedures of segmentation and convexification, a line to be used as constraint among all the possible ones must be chosen.

This can be selected according to the *maximally violated constraint*, which

3.7 Position constraints

has been successfully applied in [23]. This method defines the constraint line to be chosen as the one whose relative distance from the robot is maximum.

However, this condition can bring to the imposition of constraints that are close to robot position but related to segments that are far from it. To this aim, given the endpoints of the j -th segment \mathbf{P}_j and \mathbf{P}_{j+1} and robot position \mathbf{P}_{pos} two different situations can be considered (shown in Figure 3.12 and 3.13, respectively), taking into account the projection of robot position on the segment:

- **Projection outside the segment**

$$\text{Condition: } \overrightarrow{\mathbf{P}_j \mathbf{P}_{pos}} \cdot \overrightarrow{\mathbf{P}_j \mathbf{P}_{j+1}} \leq 0 \vee \overrightarrow{\mathbf{P}_{j+1} \mathbf{P}_{pos}} \cdot \overrightarrow{\mathbf{P}_{j+1} \mathbf{P}_j} \leq 0$$

$$\text{Closest point: } \mathbf{P}_{min} = \mathbf{P}_i \mid \min_{i \in \{j, j+1\}} d(\mathbf{P}_{pos}, \mathbf{P}_i)$$

$$\text{Associated constraint line: } \begin{cases} h_x = \mathbf{P}_{min_x} - \hat{x}_c \\ h_y = \mathbf{P}_{min_y} - \hat{y}_c \\ l = h_x \mathbf{P}_{min_x} + h_y \mathbf{P}_{min_y} \end{cases}$$

- **Projection inside the segment**

$$\text{Condition: } \overrightarrow{\mathbf{P}_j \mathbf{P}_{pos}} \cdot \overrightarrow{\mathbf{P}_j \mathbf{P}_{j+1}} \geq 0 \wedge \overrightarrow{\mathbf{P}_{j+1} \mathbf{P}_{pos}} \cdot \overrightarrow{\mathbf{P}_{j+1} \mathbf{P}_j} \geq 0$$

$$\text{Associated constraint line: } \begin{cases} h_x = \mathbf{P}_{j_y} - \mathbf{P}_{j+1_y} \\ h_y = \mathbf{P}_{j+1_x} - \mathbf{P}_{j_x} \\ l = \mathbf{P}_{j+1_x} \mathbf{P}_{j_y} - \mathbf{P}_{j_x} \mathbf{P}_{j+1_y} \end{cases}$$

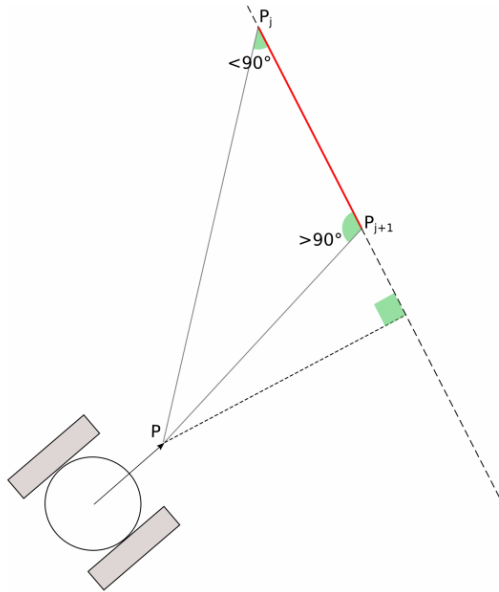


Figure 3.12: Case in which the projection on the segment is outside it.

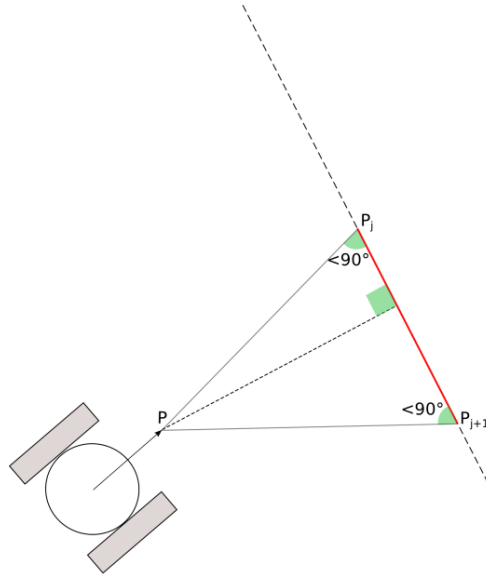


Figure 3.13: Case in which the projection on the segment is inside it.

3.8 Conclusions

In the previous sections, the system overall architecture and all the implemented constraints have been shown.

As will be seen, the usage of quadratic velocity constraints, require a high computational time and thus not suitable for implementing a system as the one described.

Furthermore, socially compliant constraints have shown to be effective in crowded environments, allowing the robot to keep a safety distance compatible with social requirements from other pedestrians. However, these constraints do not allow to effectively implement yielding behaviour, as it will be seen in Chapter 5.

Chapter 4

Trajectory modification algorithm

In the following sections, the problem of obstacle avoidance will be formalized. A novel trajectory modification algorithm (based on the novel concept of *anchor points*) that allows the implementation of the avoidance behaviour is shown.

The approach presented in this work is thought for low people density scenarios, where pedestrians will most likely have a higher speed than the wheelchair, as it will be described later. This assumption is reasonable due to the fact that for a pedestrian the average walking speed goes from $0.7 \frac{m}{s}$ to $1.15 \frac{m}{s}$, as shown in [24], while a maximum value for the longitudinal velocity for an autonomous wheelchair that takes into account safety and comfort constraints is about $0.55 \frac{m}{s}$ [13].

Finally, in section 4.1 the concept of *anchor point* is introduced, followed by the definition of the trajectory modification algorithm in section 4.2.

4.1 Anchor points

The approach presented in this work is based on the novel concept of *Anchor Points*. These are additional points that allow computing, at each time instant, an obstacle-free trajectory allowing the robot to smoothly avoid the obstacles while guaranteeing a minimum safety distance.

The solution proposed in the following is based on the assumption that the maximum allowed velocity for the robot is smaller than the one of the obstacles it encounters.

Under this assumption, it is more likely to have a situation in which the robot cannot overcome the obstacle due to its limited capabilities, as it will be shown in section 5.6.

In particular, only the following assumptions must be done in order to apply the method that will be described later:

Assumption 1 *the robot is following a generic trajectory*

$$T_{ref}(k) = \begin{bmatrix} \xi_{ref}(k) \\ \vdots \\ \xi_{ref}(k + N_{ref}) \end{bmatrix} \quad (4.1)$$

where N_{ref} is the number of reference positions computed by the Global Planner in order to reach the final position $\xi_{ref}(k + N_{ref})$;

Assumption 2 *the reference trajectory $T_{ref}(k)$ is feasible;*

Assumption 3 *the obstacle is assumed to follow a linear movement described by the line $l_v^{(j)}(k)$ and the velocity vector*

$$V_o^{(j)}(k) = \begin{bmatrix} V_{o,x}^{(j)}(k) \\ V_{o,y}^{(j)}(k) \end{bmatrix} \quad (4.2)$$

such that its position at the future time frame $k + i$ could be predicted as:

$$P_o^{(j)}(k + i) = P_o^{(j)}(k) + i\tau_s V_o^{(j)}(k) \quad (4.3)$$

where τ_s is the controller sampling time;

Assumption 4 *at the discrete time instant k the robot and the obstacle j can be represented as circles centered in $P_r(k)$, $P_o^{(j)}(k)$ with radius r_w , $r_o^{(j)}(k)$ and velocity $V_r(k)$, $V_o^{(j)}(k)$ (see Figure 4.1).*

4.1 Anchor points

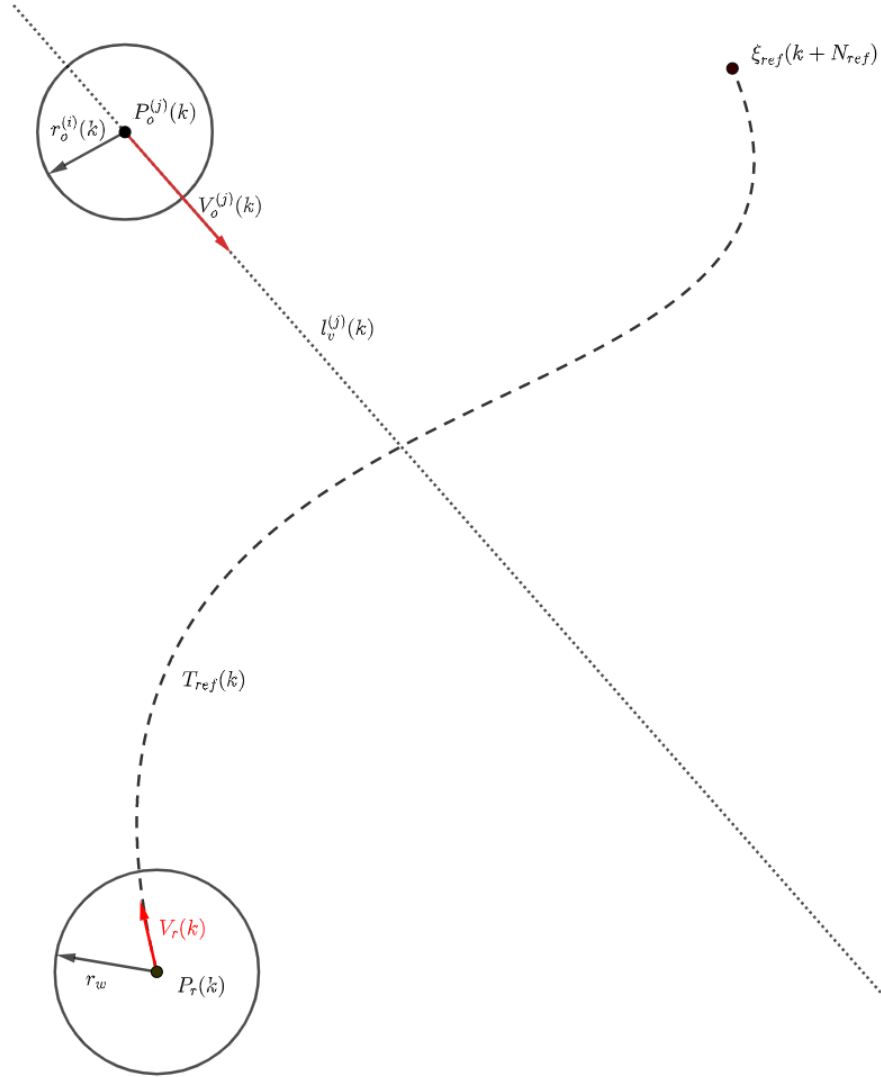


Figure 4.1: Initial situation considered, the robot at position $P_r(k)$ is following the the reference trajectory $T_{ref}(k)$ with the velocity $V_r(k)$ and an obstacle is moving along the linear trajectory $l_v^j(k)$ with velocity $V_o^j(k)$

As shown before, in order to determine an avoidance behaviour, it is possible to study the motion of the robot, approximated by a point in position $P_r(k)$ and enlarging the obstacle radius $r_o^j(k)$ to a suitable value of $d_s^j(k)$, such that a minimum safety distance could be guaranteed (see Figure 4.2).

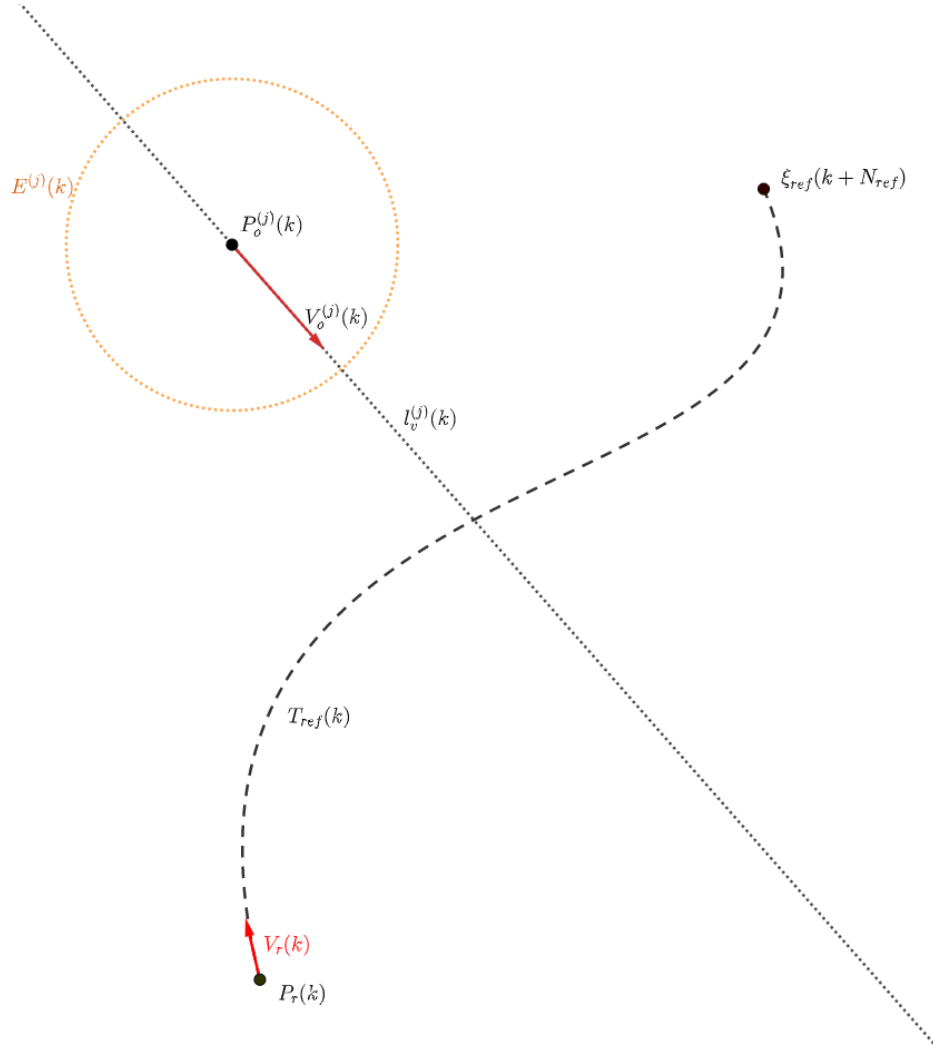


Figure 4.2: The avoidance behaviour can be guaranteed for the robot if it is guaranteed for the point $P_r(k)$ considering the obstacles enlarged by $d_s^{(j)}(k)$

In the following, the approach used for the avoidance of a single obstacle j will be introduced, while in the next section its usage in the general case with a generic set of n_{obs} obstacles that are following a linear movement will be shown.

In particular, by defining a function that allows to properly evaluate whether or not a point is in front or behind the obstacle, it is possible to formally define a condition that determines if a point in the reference trajectory is in potential collision with an obstacle.

4.1 Anchor points

To this aim, considering a vector

$$v_l = \begin{bmatrix} a_x \\ b_y \end{bmatrix} \quad (4.4)$$

and a point $P_0 \in \mathbb{R}^2$, it's possible to define the line l in \mathbb{R}^2 that is perpendicular to v_l and passes through P_0 as

$$l : a_x x + b_y y + c = 0 \quad (4.5)$$

where the coefficient c is selected such that l passes through P_0 , that is:

$$v_l^T P_0 + c = 0 \quad (4.6)$$

and, after replacing c in Eq. 4.5, the line l can be rewritten as:

$$l : v_l^T \begin{bmatrix} x \\ y \end{bmatrix} - v_l^T P_0 = 0 \quad (4.7)$$

Therefore, considering a generic point

$$P = \begin{bmatrix} P_x \\ P_y \end{bmatrix}, \quad P \in \mathbb{R}^2 \quad (4.8)$$

the distance function $d(P, l)$ between the point P and the line l is:

$$d(P, l) = \frac{|v_l^T P - v_l^T P_0|}{\sqrt{a_x^2 + b_y^2}} \quad (4.9)$$

and, recalling the fact that:

$$\|v_l\| = \sqrt{a_x^2 + b_y^2} \quad (4.10)$$

the distance function becomes:

$$d(P, l) = \frac{|v_l^T P - v_l^T P_0|}{\|v_l\|} \quad (4.11)$$

Furthermore, the function $d_\perp(P, l)$ can be defined as:

$$d_\perp(P, l) = \frac{v_l^T P - v_l^T P_0}{\|v_l\|} \quad (4.12)$$

that is, a function whose absolute value is distance $d(P, l)$ and its sign represents whether or not the point P is in the same half-plane, delimited by l , to which v_l would belong if applied in P_0 (i.e., the point is in front of or behind the point P_0 moving with velocity v_l).

This can be shown recalling the fact that the scalar product between two vectors V_1 and V_2 can be rewritten as:

$$V_1^T V_2 = \|V_1\| \|V_2\| \cos(\alpha_1 - \alpha_2) \quad (4.13)$$

where α_1 and α_2 are the angles formed by the vector V_1 and V_2 with the x positive axis taken in counterclockwise direction.

Then, by substituting this last relation in $d_\perp(P, l)$:

$$d_\perp(P, l) = \frac{\|v_l\| \|P\| \cos(\alpha_{v_l} - \alpha_P) - \|v_l\| \|P_0\| \cos(\alpha_{v_l} - \alpha_{P_0})}{\|v_l\|} \quad (4.14)$$

and, after simplifying:

$$d_\perp(P, l) = \|P\| \cos(\alpha_{v_l} - \alpha_P) - \|P_0\| \cos(\alpha_{v_l} - \alpha_{P_0}) \quad (4.15)$$

where $\|P\| \cos(\alpha_{v_l} - \alpha_P)$ and $\|P_0\| \cos(\alpha_{v_l} - \alpha_{P_0})$ are the projections of the vectors P and P_0 on the direction α_{v_l} , that is the direction perpendicular to the line l , described by the vector v_l .

This can be seen in Figure 4.3, where the distance between B and O is $\|P_0\| \cos(\alpha_{v_l} - \alpha_{P_0})$ and the one between O and A is $\|P\| \cos(\alpha_{v_l} - \alpha_P)$. Therefore, from the given definition of $d_\perp(P, l)$:

$$d_\perp(P, l) = \overline{OA} - \overline{OB} \quad (4.16)$$

that is positive when the point A is in the half plane delimited by l that would contain the vector v_l (applied in B).

4.1 Anchor points

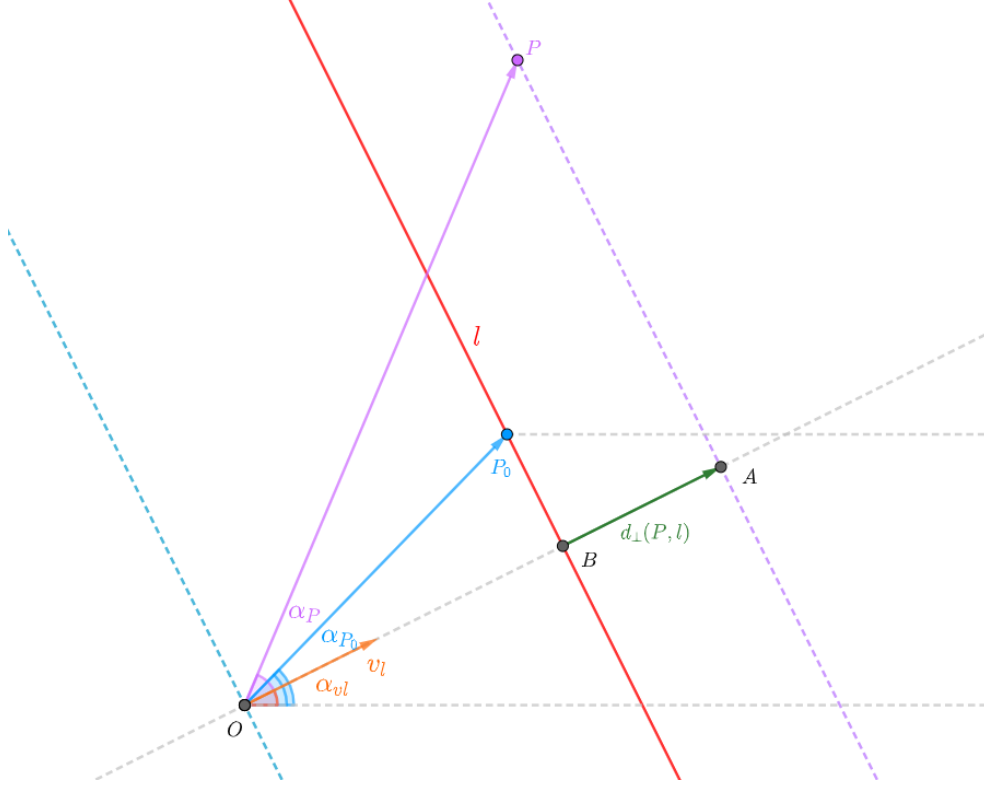


Figure 4.3: Graphical representation of $d_{\perp}(P, l)$,

As a consequence, since it is assumed that the obstacle j is moving along a linear path, it is possible to determine the set of potentially colliding points in the reference trajectory $T_{ref}(k)$, by considering the two tangents $l_s^{(j)}$ and $l_f^{(j)}$ to the enlarged obstacle $E^{(j)}$ that are parallel to its velocity (see Figure 4.4), and the line $l_b^{(j)}(k)$ that is tangent to the enlarged obstacle $E^j(k)$ and perpendicular to $V_o^{(j)}(k)$, passing through the obstacle's back $P_b^{(j)}(k)$ (see Figure 4.4) as:

$$l_b^{(j)} : V_{o,x}^{(j)}(k)x + V_{o,y}^{(j)}(k)y - V_o^{(j)}(k)^T P_b^{(j)}(k) = 0 \quad (4.17)$$

where $P_b^{(j)}(k)$ can be evaluated as the point at distance $d_s(k)$ from $P_o^{(j)}(k)$ in the direction opposite to $V_o^{(j)}(k)$:

$$P_b^{(j)}(k) = P_o^{(j)}(k) - d_s \frac{V_o^{(j)}(k)}{\|V_o^{(j)}(k)\|} \quad (4.18)$$

In that way it is possible to find the set of potentially colliding points $T_{ref}^{*(j)}$ in the reference trajectory, such that it contains all the $\xi_{ref}(k+j) \in$

$T_{ref}(k)$ that satisfy the following system:

$$\begin{cases} d(\xi_{ref}(k+j), l_v^{(j)}(k)) \leq d_s^j(k) \\ d_{\perp}(\xi_{ref}(k+j), l_b^{(j)}(k)) \geq 0 \end{cases} \quad (4.19)$$

where $j \in \{0, \dots, N_{ref}\}$.

The points in the reference trajectory that satisfy this system are the ones that:

- lie between the two tangent $l_s^{(j)}(k)$ and $l_f^{(j)}(k)$ (i.e., first inequality in Eq. 4.19);
- are in the same half-plane delimited by the back line $l_b^{(j)}(k)$, that would contain obstacle velocity $V_o^{(j)}(k)$ if applied in $P_b^{(j)}(k)$ (i.e., second inequality in Eq. 4.19).

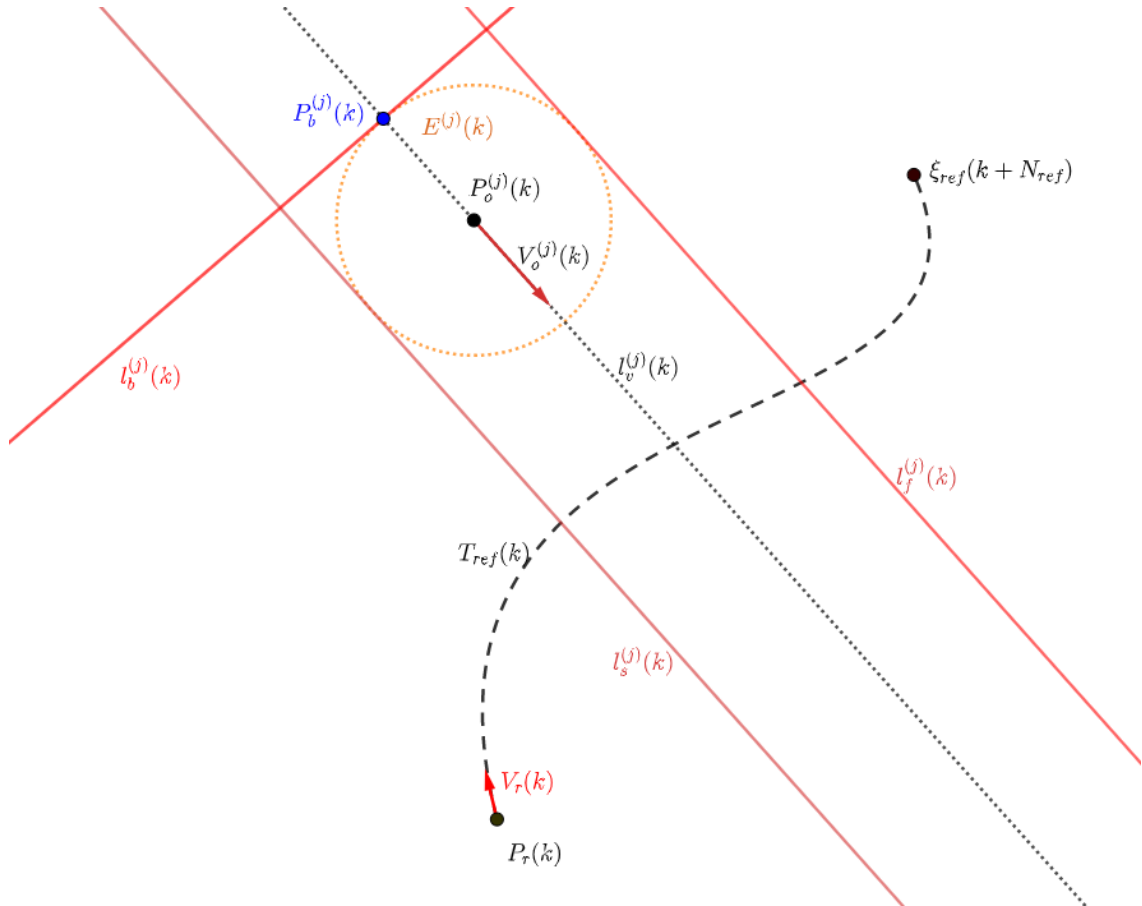


Figure 4.4: Graphical representation of the tangents to $E^{(j)}(k)$ $l_s^{(j)}(k)$ and $l_f^{(j)}(k)$ and the heading line $l_b^{(j)}(k)$ passing through obstacle's back $P_b^{(j)}(k)$

4.1 Anchor points

In addition, it can be noticed that since this method takes into account all the reference trajectory, there could exist several disjoint sets of colliding points that should correspond to different avoidance manoeuvres, one of which will be selected as the first one to be avoided, as it will be shown later.

To this aim, assuming that the reference trajectory $T_{ref}(k)$ is continuous, the set $T_{ref}^{*(j)}(k)$ can be split in n_p partitions based on the number of sections of the trajectory that intersect with obstacle's path, such that two consecutive positions in each subset $T_{ref}^{*(j,i_p)}$ correspond to consecutive positions in the original reference trajectory $T_{ref}(k)$, where i_p is the index of the partition.

Additionally, each subset $T_{ref}^{*(j,i_p)}$ can be analyzed by evaluating $\forall \xi_{ref}(k+i) \in T_{ref}^{*(j,i_p)}$ the value of

$$d(\xi_{ref}(k+i), l_b^{(j)}(k)) \quad (4.20)$$

that is, distance between the point ξ_{ref} and the line representing the back of the obstacle $l_b^{(j)}$.

In particular, in order to estimate the time $\hat{t}_{c,j}$ required by the obstacle j to collide with the reference trajectory, it's possible to find the discrete time $k_c^{(j,i_p)}$ corresponding to the first point in $T_{ref}^{*(j,i_p)}$, that is the one for which:

$$\xi_{ref}(k_c^{(j,i_p)}) \in T_{ref}^{*(j,i_p)} | \forall \xi_{ref}(k') \in T_{ref}^{*(j,i_p)} \Rightarrow k' \geq k_c^{(j,i_p)} \quad (4.21)$$

that is, the first point that has to be reached by the robot in the colliding area. Finally, recalling the fact that the obstacle is following a linear trajectory with $V_o^{(j)}(k)$, that is perpendicular to $l_b^{(j)}(k)$ by construction, the estimated time to collision $\hat{t}_{c,j}$ can be evaluated as the ratio between the distance from obstacle back line $l_b^{(j)}(k)$ to the first potentially colliding point $\xi_{ref}(k+k_c^{(j,i_p)})$ and obstacle velocity norm:

$$\hat{t}_c^{(j,i_p)} = \frac{d(\xi_{ref}(k_c^{(j,i_p)}), l_b^{(j)}(k))}{\|V_o^{(j)}(k)\|} \quad (4.22)$$

As a consequence, it is possible to evaluate among all the partitions $T_{ref}^{*(j,i_p)}$ which velocity would be required by the robot to reach the colliding area as the ratio between the total distance that has to be covered to reach $\xi_{ref}(k+k_c^{(j,i_p)})$ and the time $\hat{t}_c^{(j,i_p)}$ for which it has to be reached:

$$\|V_r^{(j,i_p)}\| = \frac{\sum_{i=k}^{k_c^{(j,i_p)}-1} d(\xi_{ref}(i), \xi_{ref}(i+1))}{\hat{t}_c^{(j,i_p)}} \quad (4.23)$$

where $\|V_r^{(j,i_p)}\|$ is the norm of the velocity required by the robot to reach the colliding area i_p associated with the obstacle j .

Notice that the value of $\|V_r^{(j,i_p)}\|$ is small either if $\hat{t}_c^{(j,i_p)}$ is big (which means that the collision is potentially far in time, and thus it could be ignored) or if the total distance computed from $\xi_{ref}(k)$ to $\xi_{ref}(k + k_c^{(j,i_p)})$ is small, which means that the obstacle is following a path such that the next positions in the reference trajectory $T_{ref}(k)$ are inside the colliding area, and thus it cannot be guaranteed to find an avoidance trajectory.

Finally, among all the partitions $T_{ref}^{*(j,i_p)}$ that are associated with a suitable value of $\|V_r^{(j,i_p)}\|$ it is possible to identify which one is associated with the smallest $\hat{t}_c^{(j,i_p)}$ and consider it for the avoidance, as it will be the first one to be encountered.

In order to properly modify the reference trajectory with another one that guarantees the avoidance of the obstacle, a trajectory that goes behind the obstacle (considering its velocity at each discrete time instant) will be planned.

To this aim, the anchor point $P_A^{(j)}(k)$ (that is, a point that can be considered *safe* with respect to obstacle j at current time instant k) can be found as the intersection between the tangent from the robot position $P_r(k)$ to the obstacle enlarged circle $E^j(k)$ and the obstacle back line $l_b^{(j)}(k)$, where among the two possible tangent lines, the one that satisfies the condition:

$$d_{\perp}(P_A^{(j)}(k), l_b^{(j)}(k)) \leq 0 \quad (4.24)$$

is taken (see Figure 4.5), that is, the one that is contained in the half-plane delimited by $l_b^{(j)}(k)$ and would not contain obstacle velocity if applied in $P_b^{(j)}(k)$.

Notice that, if current robot position does not belong to the colliding area, the anchor point $P_A^{(j)}(k)$ always exists.

4.1 Anchor points

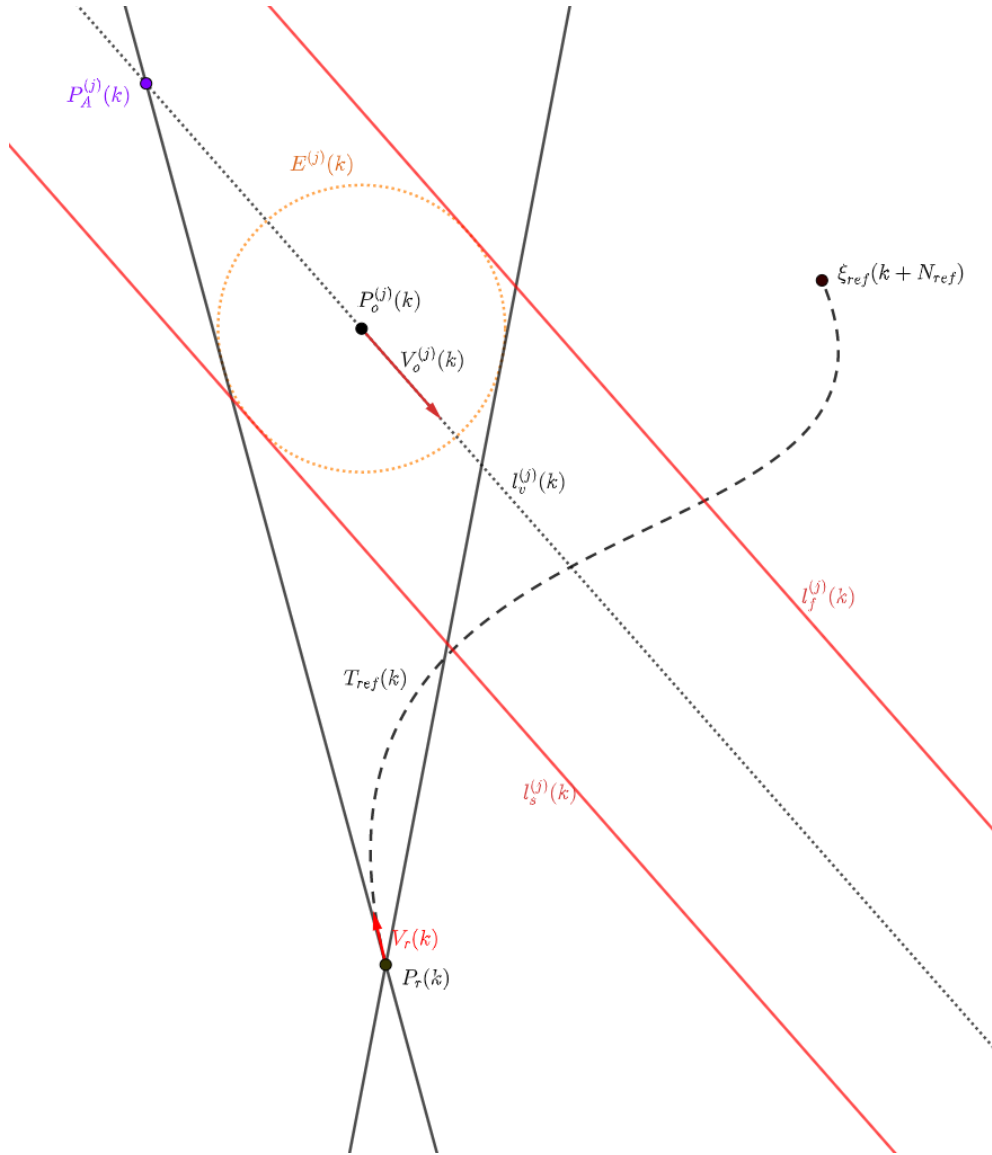


Figure 4.5: Graphical representation of the anchor point $P_A^{(j)}(k)$ computed for the obstacle j at the discrete time instant k

The point $P_A^{(j)}(k)$ is, by construction, a point behind the obstacle j at the discrete-time instant k , which cannot be reached by it at any future time instant. In fact, the *anchor point* is, by construction, a point that would require a velocity outside of the Collision Cone (described in section 2.2.3) to be reached.

In addition, in order to reach the *anchor point* by guaranteeing the required safety distance from the obstacle, two points in correspondence of the boundary lines $l_s^{(j)}(k)$ and $l_f^{(j)}(k)$ can be chosen to be followed with a velocity that is perpendicular to the obstacle's one.

4.1 Anchor points

- the first one, $\hat{L}_s^{(j)}(k)$, from $P_r(k)$ to $P_{A,s}^{(j)}(k)$;
- the second one, $\hat{L}^{(j)}(k)$, from $P_{A,s}^{(j)}(k)$ to $P_{A,f}^{(j)}(k)$;
- the third one, $\hat{L}_f^{(j)}(k)$, reconnecting the path from $P_{A,f}^{(j)}(k)$ to a *merging* point $P_f^{(j)}(k)$ in the original reference trajectory $T_{ref}(k)$.

The choice of the point $P_f^{(j)}(k)$ is not trivial, however assuming that the reference trajectory obtained by the Global Planner is built in such a way that it reflects the presence of obstacles in the map, the first point $\xi_{ref}(k_f^{(j)})$ on the reference trajectory that satisfies the conditions

$$dist(\xi_{ref}(k_f^{(j)}), \xi_{ref}(k_c^{(j,i_p)} + n_c^{(j,i_p)})) - r_w \geq 0 \quad (4.26)$$

where $dist(\cdot, \cdot)$ is the Euclidean distance function and

$$k_f^{(j)} > k_c^{(j,i_p)} + n_c^{(j,i_p)} \quad (4.27)$$

can be selected as $P_f^{(j)}(k)$, where $k_c^{(j,i_p)}$ is the discrete time instant at which corresponds the first element of the set of potentially colliding points $T^{*(j,i_p)}$ in the reference trajectory with the obstacle j at the considered partition i_p (i.e., the one that has to be avoided), $n_c^{(j,i_p)}$ the cardinality of the set $T^{*(j,i_p)}$ and $k_f^{(j)}$ the discrete time instant in the reference trajectory at which $\xi_{ref}(k_f^{(j)}) = P_f^{(j)}(k)$.

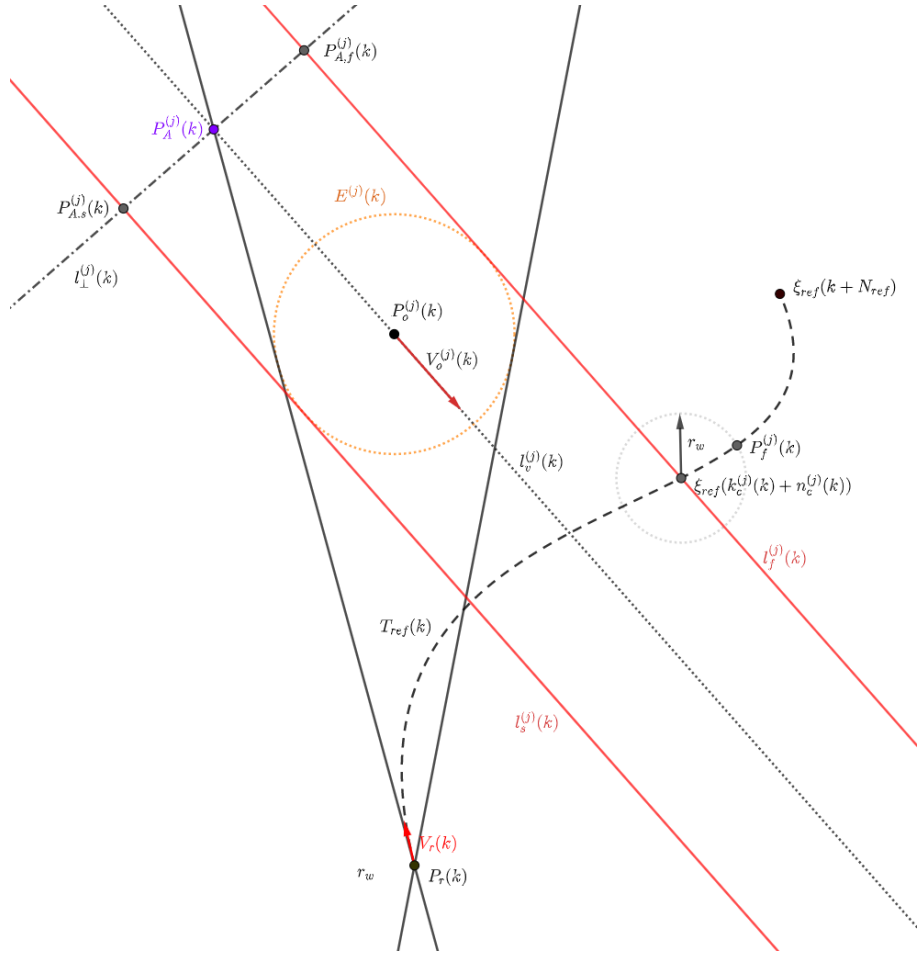


Figure 4.7: Graphical representation of the point $P_f^{(j)}(k)$, that is the closest one to $P_{A,f}^{(j)}(k)$ belonging to the reference trajectory $T_{ref}(k)$

In that way, the avoidance trajectory will merge in the reference trajectory in the first point possible, taking into account an additional space that would be occupied by the wheelchair while going back into the trajectory $T_{ref}(k)$.

In fact, the choice of a point like the closest point to $P_{A,f}^{(j)}(k)$ (i.e., the one that minimizes the space required to perform the reconnecting manoeuvre) belonging to the reference trajectory $T_{ref}(k)$ would mean that the computed avoidance trajectory that connects $\xi_{ref}(k)$ to $\xi_{ref}(k_f^{(j)})$ passing by the anchor point could be a feasible one. However, assuming that the global planning algorithm will plan a nearly optimal (or even an optimal one) trajectory based on map constraints, it is reasonable to think about the fact that if the global planner hasn't selected this trajectory, it's because it isn't feasible.

Finally, in order to evaluate if there is a potential collision with the obstacle and the reference trajectory during the avoidance maneuver, the reference trajectory will be modified at each iteration by computing a linear trajectory $T_l(k)$ directly connecting $P_r(k)$ and $P_f^{(j)}(k)$ (see Figure 4.9).

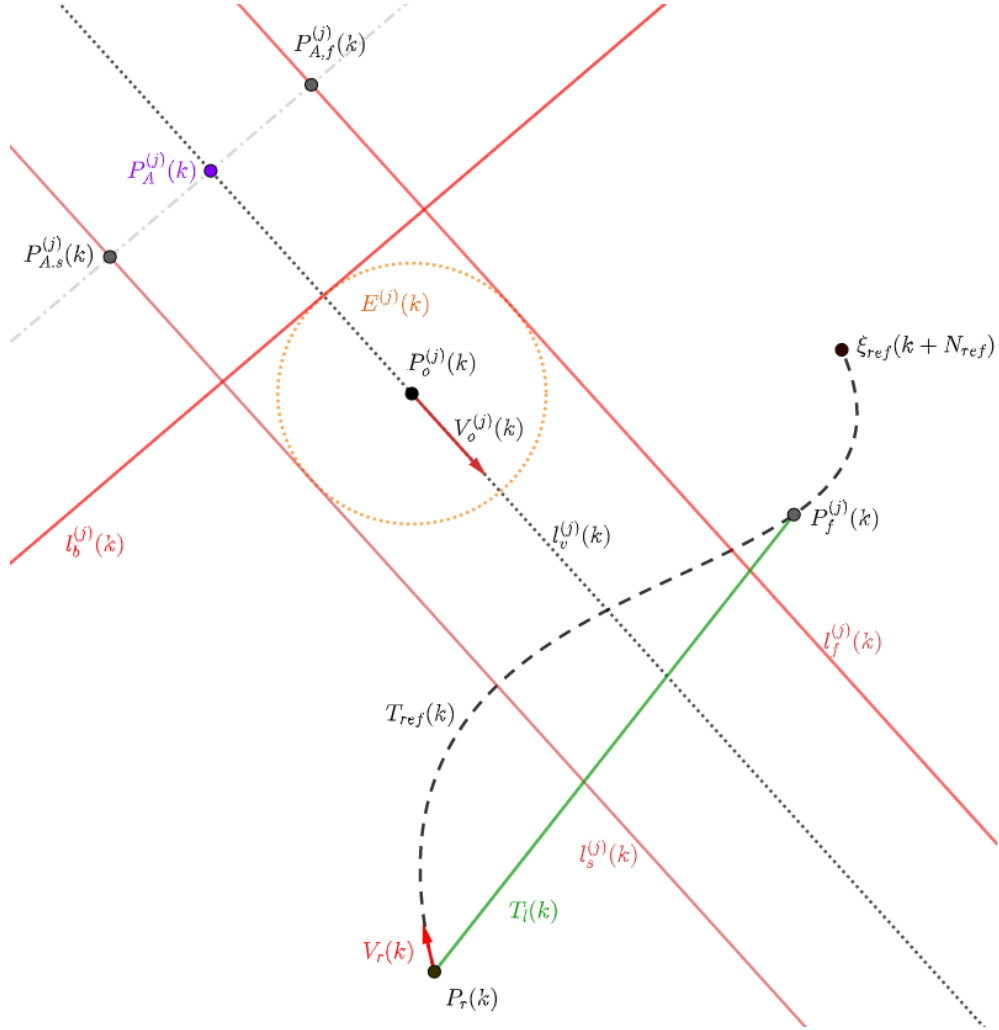


Figure 4.9: Graphical representation of the merging trajectory $T_l(k)$

Notice that, after verifying if both $T_A^{(j)}(k)$ and $T_l(k)$ are collision-free according to the map, which can be done by suitably enlarging the planned positions in order to take into account robot dimensions, it is possible to follow the new trajectory (which includes the avoidance trajectory) in order to avoid the obstacle.

4.2 Anchor points algorithm definition

In order to exploit the technique presented in the previous section in a generic situation where it could be required to avoid different obstacles at the same time, the concept of *reference point* must be introduced. In particular, the term *reference point* will refer to all the points belonging to the considered trajectory that correspond to:

- robot position;
- an anchor point $P_A^{(j)}(k)$;
- a merging point $P_f^{(j)}(k)$;

in fact, these points are the ones that could be considered *safe* along the new trajectory obtained after the trajectory modification procedure, and can be used to compute the avoidance trajectory for multiple obstacles. Before going into the definition of the algorithm, some auxiliary functions are defined:

- *GetPotentialCollision*: a function that takes in input a trajectory and returns the first potential collision according to the technique shown in the previous section;
- *GetClosestRefPoint*: a function that takes in input a potentially colliding obstacle and evaluates the closest *reference point* on the given trajectory that comes before the potential collision;
- *GetAnchorPoint*: a function that takes in input an obstacle and evaluates the position of the *anchor point* for the given reference point and obstacle;
- *GetAvoidanceTrajectory*: a method that takes in input an *anchor point* and an obstacle and evaluates the avoidance trajectory from the provided reference point;
- *IsFreeMap*: a function that takes in input a trajectory and the *Map*, evaluating if it has no collisions inside the map;
- *MergeTrajectories*: a function that takes in input two trajectories and merges them opportunely;
- *MergeAvoidanceTrajectory*: a function that takes in input a reference trajectory and an avoidance one and merges them;

- *GetMergingPoint*: a function that returns the closest merging point $P_f(k)$ to current robot position;
- *ComputeLinearTrajectory*: a function that computes a linear trajectory between two given positions.

Finally, the algorithm can be defined as follows, at each discrete time instant k :

Algorithm 2: Trajectory modification

```

1  stop  $\leftarrow$  false;
2  AvoidedObs  $\leftarrow$   $\emptyset$ ;
3   $T_{new}(k) \leftarrow T_{ref}(k)$ ;
4  while  $\neg stop$  do
5       $Obs(k) \leftarrow GetPotentialCollision(T_{new}(k))$ ;
6      if  $Obs(k) \neq \emptyset \wedge Obs(k) \notin AvoidedObs$  then
7           $P_{ref}(k) \leftarrow GetClosestRefPoint(Obs(k), T_{new}(k))$ ;
8           $P_A(k) \leftarrow GetAnchorPoint(P_{ref}(k), Obs(k))$ ;
9           $T_A(k) \leftarrow$ 
             $GetAvoidanceTrajectory(P_{ref}(k), P_A^{(j)}(k), Obs(k))$ ;
10          $AvoidedObs = [AvoidedObs, Obs(k)]$ ;
11         if IsFreeMap( $T_A, Map$ ) then
12              $T_{new}(k) \leftarrow$ 
                 $MergeAvoidanceTrajectory(T_{new}(k), T_A^{(j)}(k))$ ;
13         else
14              $stop \leftarrow true$ ;
15         end
16     else
17          $stop \leftarrow true$ ;
18     end
19 end
20  $P_f^*(k) \leftarrow GetMergingPoint()$ ;
21  $T_l(k) \leftarrow ComputeLinearTrajectory(P_r(k), P_f^*(k))$ ;
22 if IsFreeMap( $T_l(k), Map$ ) then
23      $T_{ref}(k) \leftarrow MergeTrajectories(T_{ref}(k), T_l(k))$ ;
24 end
25 return  $T_{new}(k), T_{ref}(k)$ ;

```

From a practical point of view, the algorithm will evaluate and merge the avoidance trajectory into the reference one until one of the exit conditions is met. The exit conditions are satisfied when either the computed

4.2 Anchor points algorithm definition

trajectory $T_{new}(k)$ does not potentially collide with any obstacle or it does collide with an obstacle that should have been precedently avoided (see **Line 6**), or when the avoidance trajectory $T_A^{(j)}(k)$ would collide according to the map (see **Line 11**); in that way, the while loop will stop after maximum n_{obs} iteration. Notice that the position of the *anchor point*, as it has been defined in the previous section, depends on the current position of the obstacle and not considers the future ones, despite taking a point that is not directly reachable by the obstacle with its current velocity direction. In fact, it is not possible to determine a priori which velocities the optimizer will select according to the set of constraints. Thus a point that could be reachable with any speed is selected (i.e., the *anchor point*). Another thing worth mentioning is that, due to the previous considerations, at each iteration, the *anchor point* corresponding to a specific obstacle will move with the obstacle. Then, the robot will adjust its trajectory accordingly until merging again in the original reference one. Finally, after completing the while loop, the algorithm will return the trajectory $T_{new}(k)$ that will be used for solving the MPC problem (since it contains the avoidance trajectory), and the new reference trajectory $T_{ref}(k)$ that will be used to evaluate potential collisions at next iterations, and, eventually, will be followed until reaching again the original reference trajectory if there will be no more potential collisions.

Chapter 5

Results

In the following sections, the results obtained in the implementation of the velocity constraints introduced in section 3.5 and the avoidance algorithm presented in section 4.2 are shown.

In particular, in section 5.3, the results obtained in simulating the quadratic velocity constraints discussed in section 3.5.1 are presented. However, these constraints affect the computational time significantly, and thus the linear velocity constraints defined in section 3.5.2 are more suitable for the implementation in a linear MPC controller, as shown in section 5.4. Finally, in sections 5.6 and 5.7, the results obtained in simulating different cases for the novel approach presented in section 4.2 will be shown.

5.1 Simulation environment

The simulations done in this work have been achieved by implementing the required functionalities in the system described in section 3.1, which has been implemented as goal regulation instead of reference tracking. Thus the simulations done in sections 5.3 and 5.4 will be defined by a set of goal positions that the robot has to reach.

Moreover, simulations were run on a personal computer equipped with an Intel Core i7 4650U (3.3 GHz) CPU and 8 GB of RAM (DDR3). During the simulation, both logs and ROS *topics* have been recorded to be analyzed on MatLab.

To be specific, the pedestrian movement has been emulated by the library *Pedsim*. At the same time, the simulation of data perceived by laser scanners is handled by *ROS Stage* (a standard ROS node for the simulation of mobile robots navigation). These, in turn, are used by the library

find_moving_objects to detect and estimate obstacle motion.

5.2 Model parameters

Regarding the wheelchair model presented in section 3.3, the cost function introduced has the form:

$$J(k) = \sum_{i=0}^{N-1} (\|\xi(k+i) - \xi_{ref}(k+i)\|_Q^2 + \|u(k+i)\|_R^2) + \|\xi(k+N) - \xi_{ref}(k+N)\|_S^2 \quad (5.1)$$

with the weight matrices

$$Q = \begin{bmatrix} q & 0 \\ 0 & q \end{bmatrix} \quad R = \begin{bmatrix} r & 0 \\ 0 & r \end{bmatrix} \quad S = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \quad (5.2)$$

where Q and R must be tuned experimentally to ensure reference tracking and control performance, and S must be selected to guarantee the asymptotic stability of the system, as described in section 3.4.

In particular, with reference to the work done in [8], the following choices for Q , R and S have gained acceptable results:

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad R = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix} \quad S = \begin{bmatrix} 43 & 0 \\ 0 & 43 \end{bmatrix} \quad (5.3)$$

And, to avoid requiring too much computational time for the solution of the MPC optimization problem, the horizon length $N = 15$ and controller sampling time $\tau_s = 0.2[s]$ have been selected.

Finally, recalling the introduction of n_{sl} slack variables in the cost function:

$$\bar{J}(k) = J(k) + \|u_{sl_p}(k)\|_{S_p}^2 \quad (5.4)$$

with

$$S_p = \begin{bmatrix} s_p & & \\ & \ddots & \\ & & s_p \end{bmatrix}_{(n_{sl}, n_{sl})} \quad (5.5)$$

an acceptable value for s_p is:

$$s_p = 10^9 \quad (5.6)$$

5.3 Quadratic velocity constraints

The quadratic velocity constraints introduced in section 3.5.1, represent the following condition $\forall i \in \{0, \dots, N - 1\}$:

$$0 \leq \sqrt{v_{P_x}^2(k+i) + v_{P_y}^2(k+i)} \leq v_{max} \quad (5.7)$$

where $v_{max} = 0.55[\frac{m}{s}]$.

The implementation of these constraints has been successfully achieved. However, it required to set the iLog CPLEX solver to *BARRIER* (instead of *DUAL*) due to the nonlinearity of the constraints.

In order to show that the results provided by the optimizer have been verified, starting from a situation like the one depicted in Figure 5.1, the robot started moving at $t = 9.1[s]$ following the trajectory described by the goals represented by the green circles. Additionally, during its motion, the robot will detect the presence of a fixed obstacle (corresponding to a wall) at $y = -1.5$.

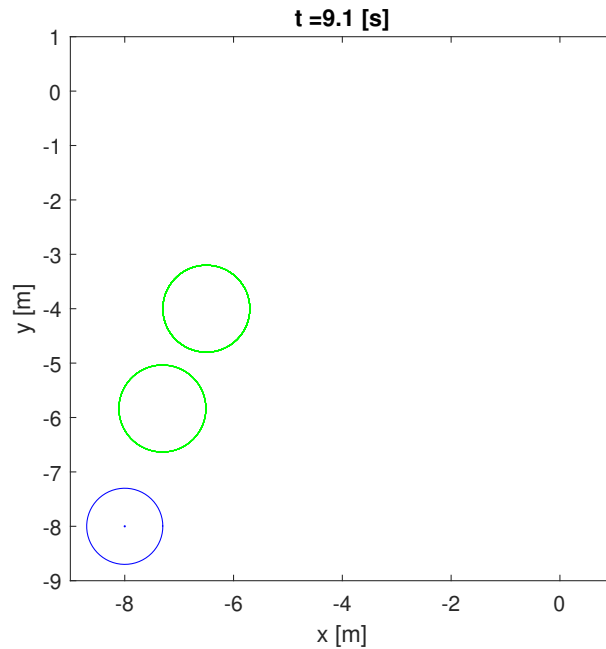


Figure 5.1: Initial situation in which the robot (in blue) is asked to follow the path defined by the two goals represented in green

The robot has reached the final position at $t = 45[s]$ following the trajectory shown in Figure 5.2 and 5.3 with the velocity profile in Figure 5.4,

where on the vertical axis is reported the speed value in [$\frac{m}{s}$] and on the horizontal one, the time in [s].

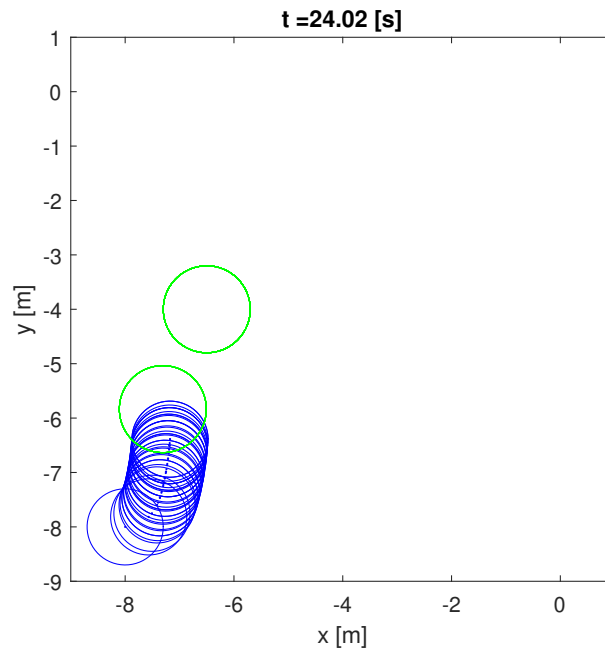


Figure 5.2: Actual trajectory followed by the robot to reach first goal, with both quadratic and linear constraints

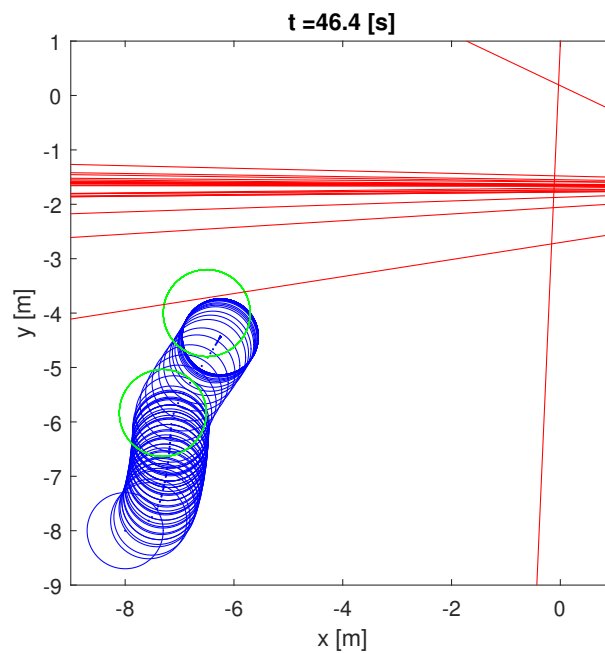


Figure 5.3: Actual trajectory followed by the robot to reach the final goal, with both quadratic and linear constraints.

5.3 Quadratic velocity constraints

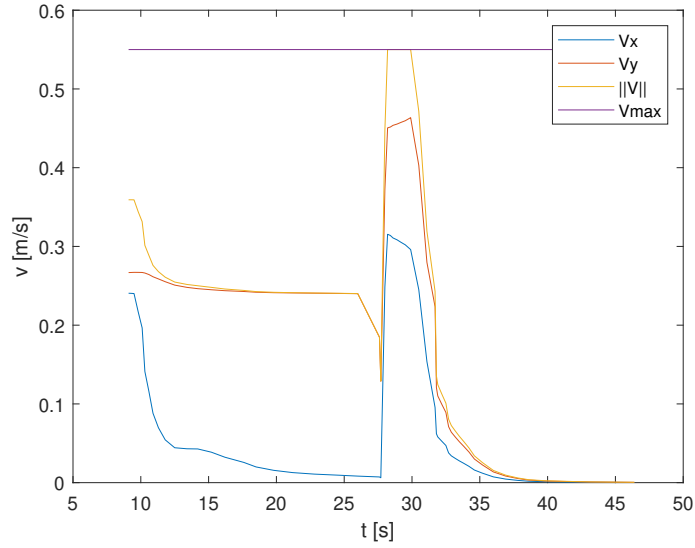


Figure 5.4: Velocity profile with quadratic constraints. The blue and orange lines represent v_{P_x} and v_{P_y} , the yellow line represents $\|v_P\|$ and the purple one the boundary value v_{max} .

Although the results show that the constraint is correctly fulfilled, the computational time increases significantly with nonlinear constraints. This can be seen in Figure 5.5 where on the vertical axis the time required by the control loop t_c in [s] is reported, and on the horizontal one the absolute time t [s].

Figure 5.5: Loop time profile with quadratic constraints and *BARRIER* solver

In summary, this last simulation had the following characteristics:

- *BARRIER* solver;
- quadratic constraints on maximum velocity;
- linear velocity variation constraints;
- linear position constraints corresponding to a wall.

In order to better analyze the time profile obtained, the same simulation has been done under different configurations. At first, linear constraints has been removed, which means that the test has been carried out with:

- *BARRIER* solver;
- quadratic constraints on maximum velocity.

In that case, the robot has followed the trajectory shown in Figure 5.6 and 5.7 with the velocity profile shown in Figure 5.8.

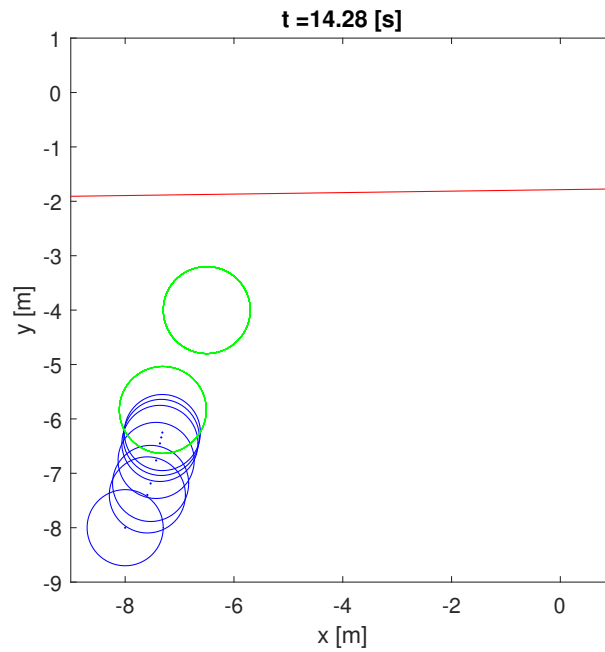


Figure 5.6: Actual trajectory followed by the robot to reach the first goal, with only quadratic constraints.

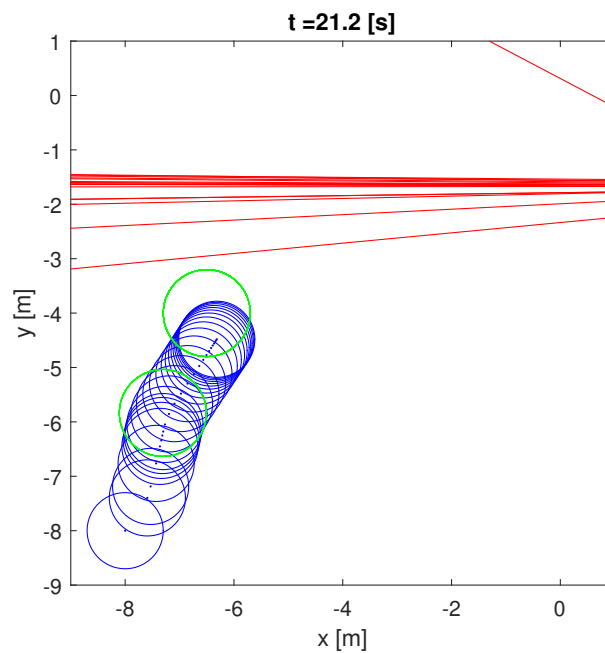


Figure 5.7: Actual trajectory followed by the robot to reach the final goal, with only quadratic constraints.

5.3 Quadratic velocity constraints

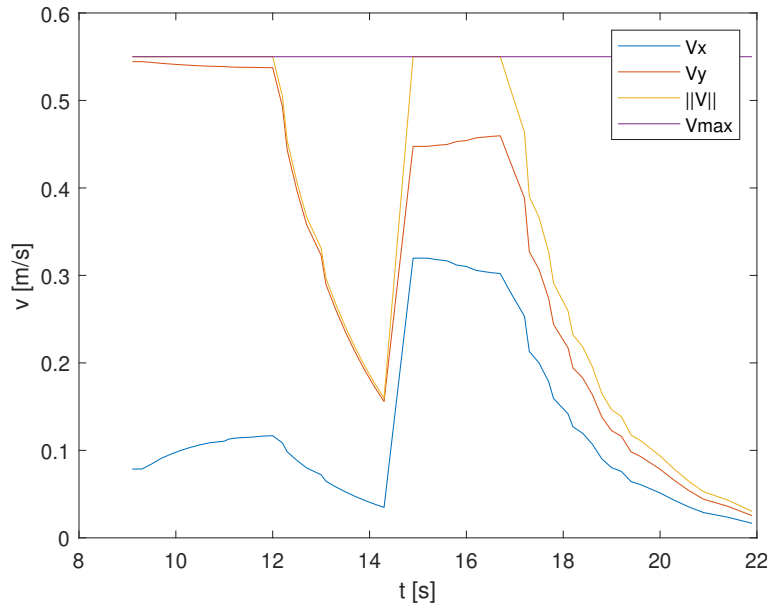


Figure 5.8: Velocity profile with only quadratic constraints.

In particular, it can be noticed that due to the absence of velocity variation constraints, the robot started moving at the maximum allowed velocity. Additionally, the loop time profile obtained in that situation can be seen in Figure 5.9.

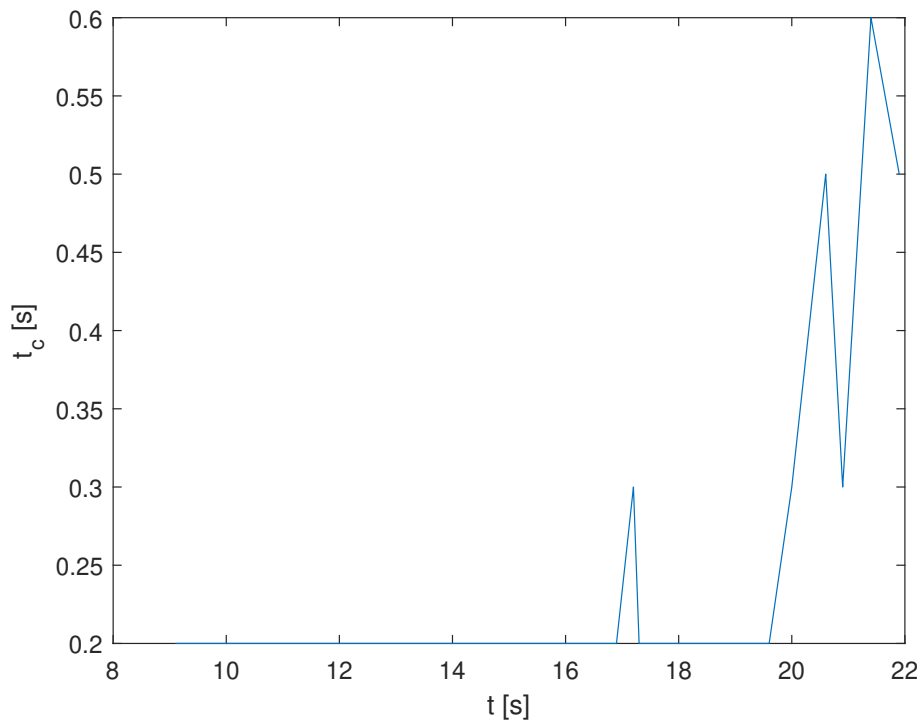


Figure 5.9: Loop time profile with only quadratic constraints.

However, the time profile obtained shows that even if it exceeds the loop time limit of 0.2[s], it does not reach the levels obtained in the original simulation.

Therefore, the following configuration has been set up:

- *BARRIER* solver;
- linear velocity variation constraints;
- linear position constraints, reflecting the presence of a wall.

In that case, the final trajectory has been the one shown in Figure 5.10 and 5.11, which has been followed with the velocity profile shown in Figure 5.12.

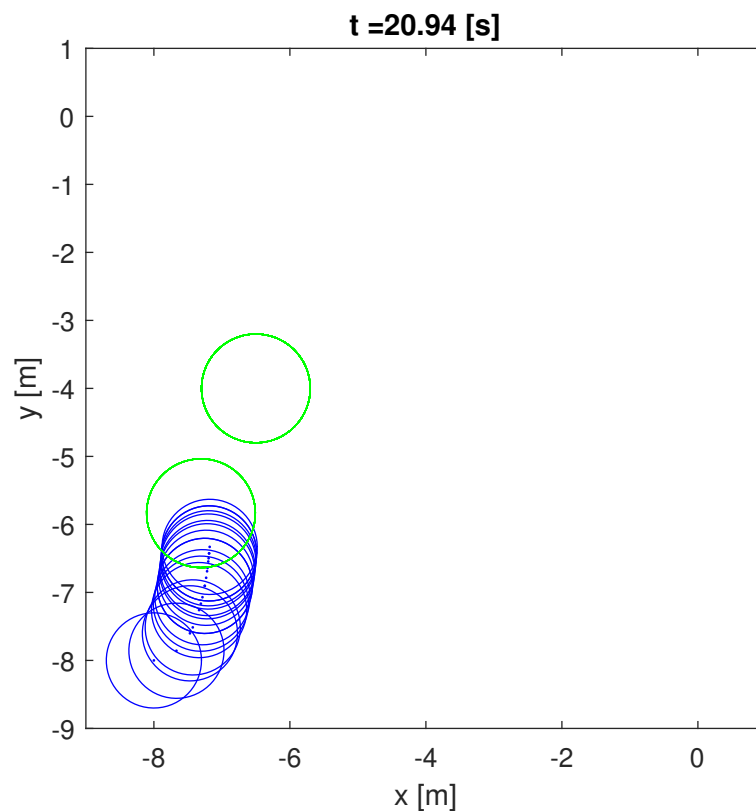


Figure 5.10: Actual trajectory followed by the robot to reach the first goal, with only linear constraints and *BARRIER* solver.

5.3 Quadratic velocity constraints

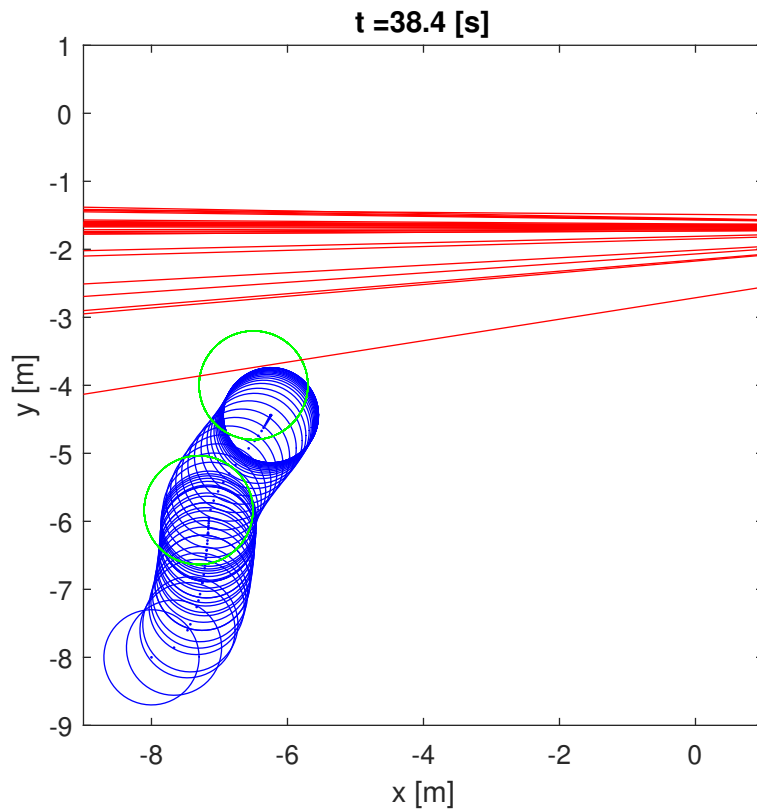


Figure 5.11: Actual trajectory followed by the robot to reach the final goal, with only linear constraints and *BARRIER* solver.

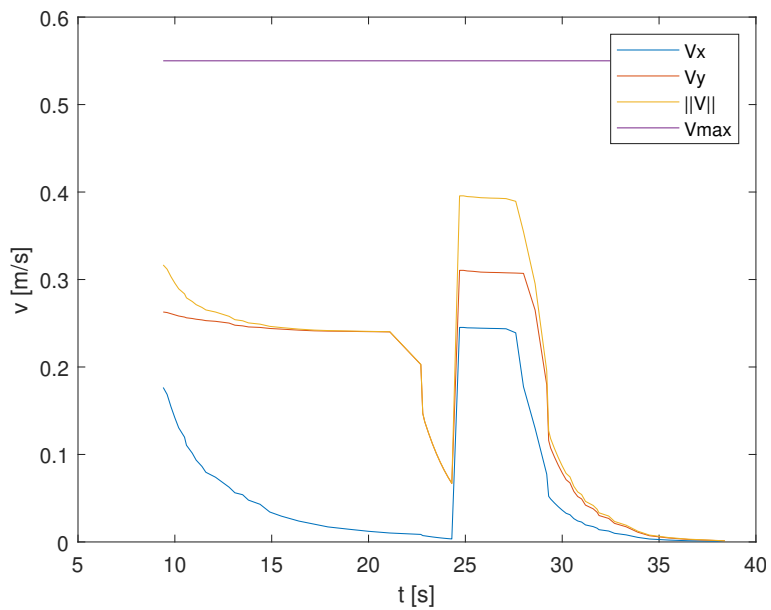


Figure 5.12: Velocity profile with only linear constraints and *BARRIER* solver

Additionally, the loop time profile has shown a behaviour similar to the one obtained in the original simulation case (see Figure 5.13).

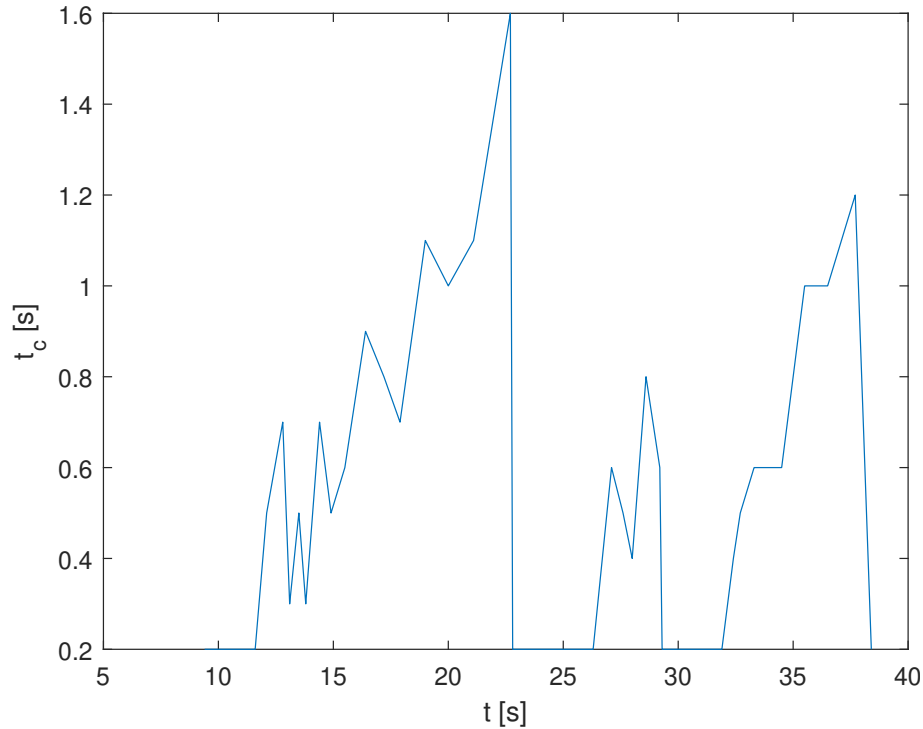


Figure 5.13: Loop time profile with only linear constraints and *BARRIER* solver

Furthermore, to check if the linear constraints imposed in that case are correct and the problem is solvable, this last case has been tested using the *DUAL* optimizer, which solves the dual problem in order to determine faster an optimal solution to the problem expressed in Eq. 3.1, that is:

- *DUAL* solver;
- linear velocity variation constraints;
- linear position constraints, corresponding to a wall.

In that situation, the robot has followed the trajectory shown in Figure 5.15, with the velocity profile shown in Figure 5.16. Notice that the robot has exceeded the maximum allowed velocity due to the absence of constraints on it during this test case.

5.3 Quadratic velocity constraints

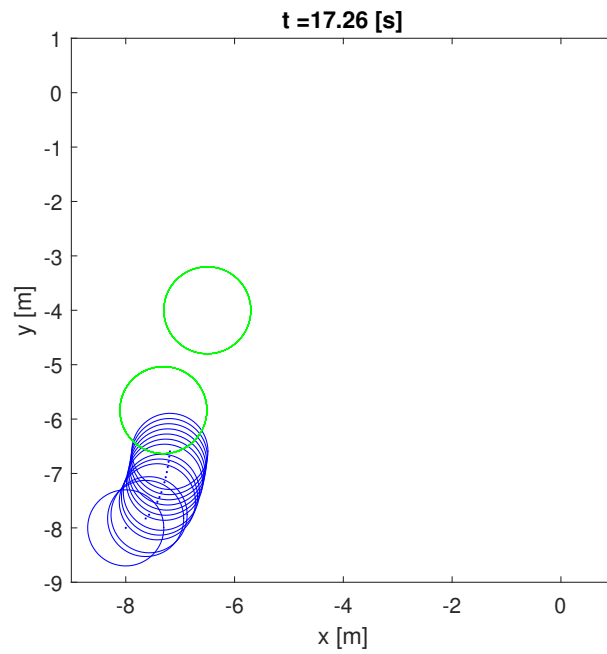


Figure 5.14: Actual trajectory followed by the robot to reach the first goal, with only linear constraints and *DUAL* solver

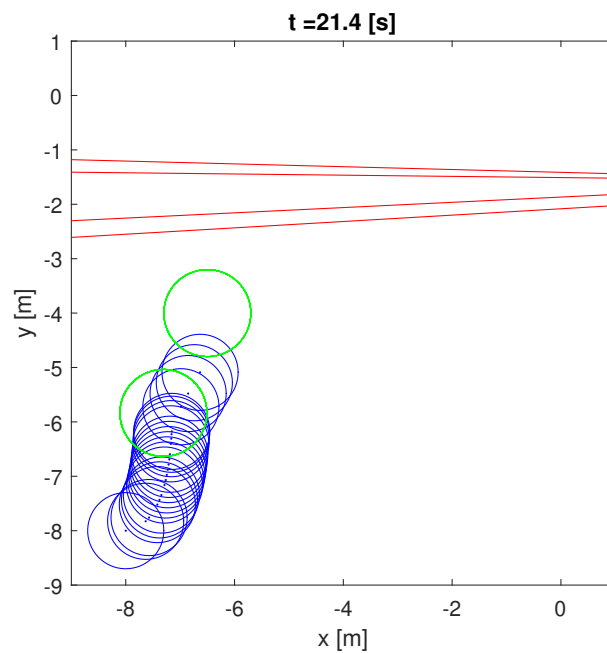


Figure 5.15: Actual trajectory followed by the robot to reach the final goal, with only linear constraints and *DUAL* solver

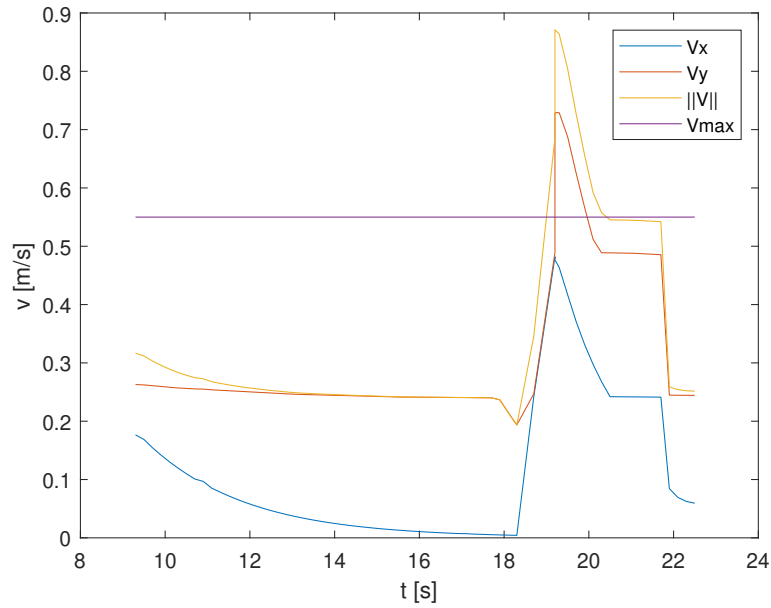


Figure 5.16: Velocity profile with only linear constraints. and *DUAL* solver

In particular, the loop time profile in that case has not shown significant variation, except for one iteration that required $0.3[s]$ instead of the default $\tau_s = 0.2[s]$.

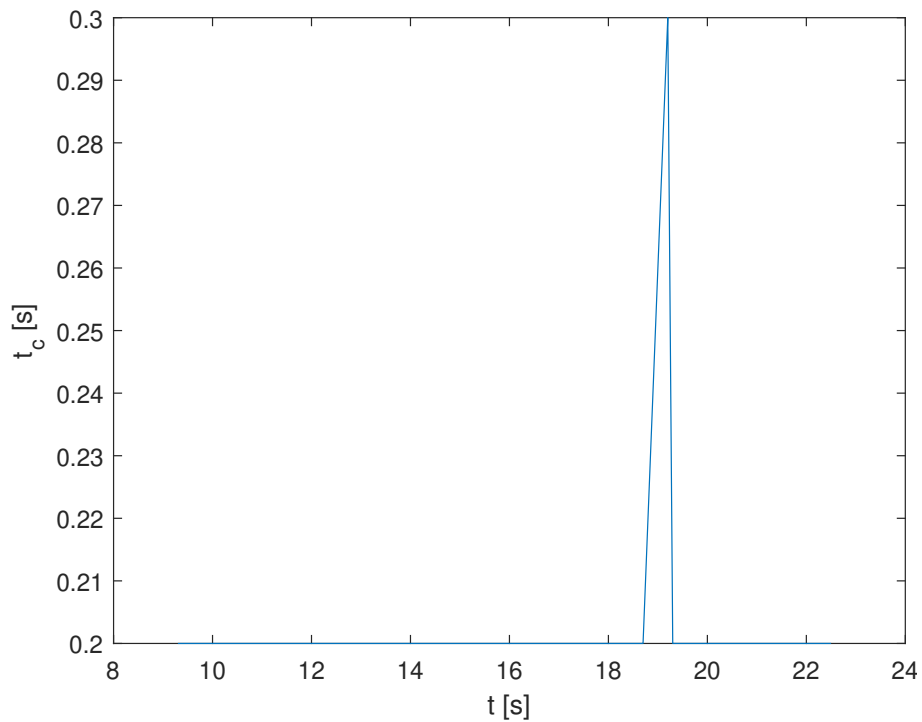


Figure 5.17: Loop time profile with only linear constraints and *DUAL* solver

5.4 Linear velocity constraints

Therefore, by comparing the time profile in Figure 5.17 with the trajectory in Figure 5.14, that higher loop times occur when the robot is close to the goal considered in the current prediction horizon. That effect is more evident in the cases where the *BARRIER* solver has been used, as it can be seen by comparing Figures 5.2, 5.10 and 5.10 with the loop time profiles in Figures 5.5, 5.9 and 5.13 respectively.

5.4 Linear velocity constraints

The linear constraints introduced in section 3.5.2 assumes that the autonomous wheelchair can be described as a differential drive robot, which requires the definition of the wheel radius r_{wheel} and the distance between wheels d , from which it is possible to evaluate:

$$\bar{\omega}_M = \frac{v_{max}}{r_{wheel}} \quad (5.8)$$

and

$$\bar{\omega}_m = \frac{v_{min}}{r_{wheel}} \quad (5.9)$$

where $v_{max} = 0.55[\frac{m}{s}]$ and $v_{min} = -0.55[\frac{m}{s}]$. Finally, with reference to the experimental wheelchair introduced in section 3.3, $r_{wheel} = 0.36[m]$ and $d = 0.65[m]$, which can be used to evaluate the angular velocity of the wheels ω_R and ω_L using the relation in Eq. 3.51 and to check if the following conditions are verified:

$$\begin{aligned} \bar{\omega}_m &\leq \omega_R \leq \bar{\omega}_M \\ \bar{\omega}_m &\leq \omega_L \leq \bar{\omega}_M \end{aligned} \quad (5.10)$$

Due to the linearity of these constraints, computational time is not affected and, running the same simulation case used for quadratic velocity constraints, they have shown to behave correctly, as it can be seen in Figure 5.18 where on the vertical axis the value of the angular speed in $[\frac{rad}{s}]$ is reported and on the horizontal one the time in $[s]$.

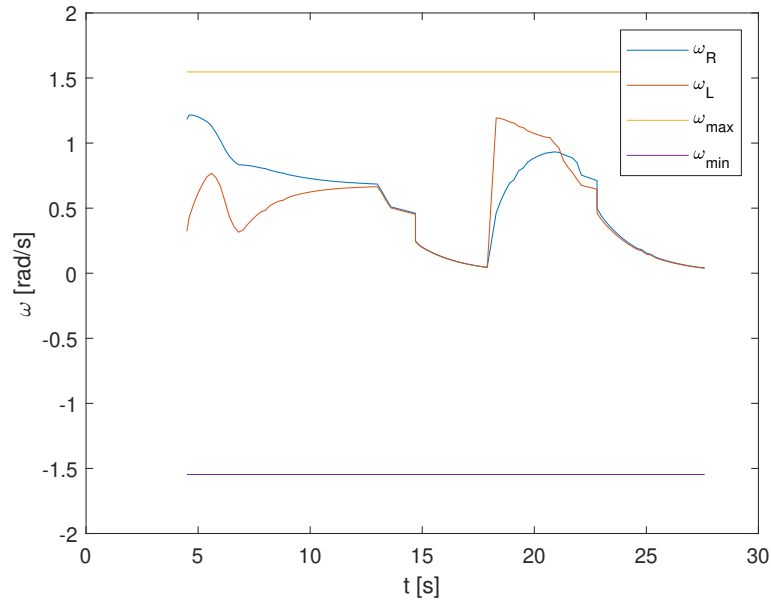


Figure 5.18: Angular velocity profile with linear constraints, the blue and orange line represent ω_L and ω_R while the purple and yellow lines represent the boundary values $\bar{\omega}_m$ and $\bar{\omega}_M$.

5.5 Trajectory modification algorithm

In the following, the behaviour of the solution proposed in this work will be shown, where the algorithm presented in section 4.2 has been implemented on *MATLAB* in a reference tracking context (as described in section 3.2).

In particular, with reference to [7], the MPC problem is characterized by the following weight matrices:

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad R = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix} \quad S = \begin{bmatrix} 9.67 & 0 \\ 0 & 9.67 \end{bmatrix} \quad (5.11)$$

with a prediction horizon of $N_h = 20$ and a slack variables weight of

$$s_p = 10^9 \quad (5.12)$$

Considering a simple case like the one shown in Figure 5.19, where the robot (the blue circle) has to follow the reference trajectory (the blue path), while an obstacle characterized by:

$$P_{obs}^{(1)} = \begin{bmatrix} -1 \\ 11 \end{bmatrix}, \quad V_{obs}^{(1)} = \begin{bmatrix} 0.3 \\ -0.4 \end{bmatrix} \quad (5.13)$$

5.5 Trajectory modification algorithm

is in potential collision with the robot.

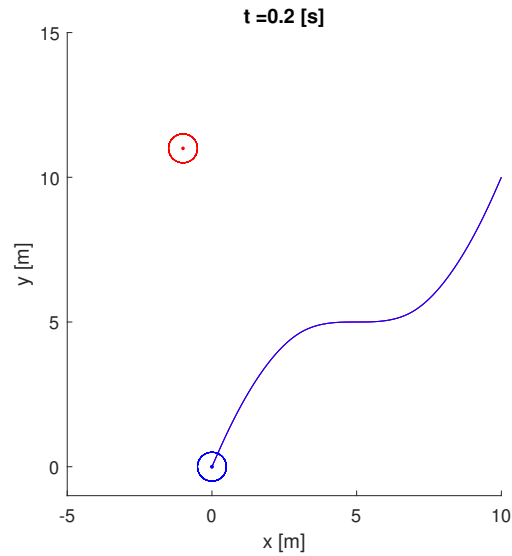


Figure 5.19: Initial simple avoidance situation

Due to the presence of a potential collision between the obstacle and the reference trajectory, the trajectory modification algorithm will compute the avoidance trajectory as described in Chapter 4.2 and shown in Figure 5.20.

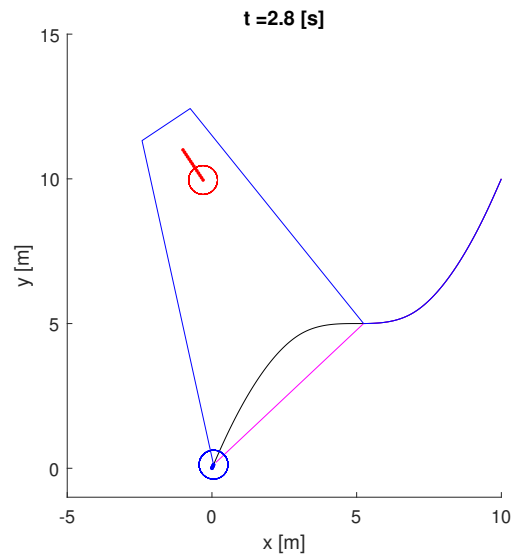


Figure 5.20: When in a potential collision, the robot will plan the avoidance trajectory in blue, while considering the magenta trajectory as a reference in next iterations

Therefore, the robot will start steering and following the avoidance trajectory, which will vary at each considered time instant, as can be observed in Figures 5.21 and 5.22.

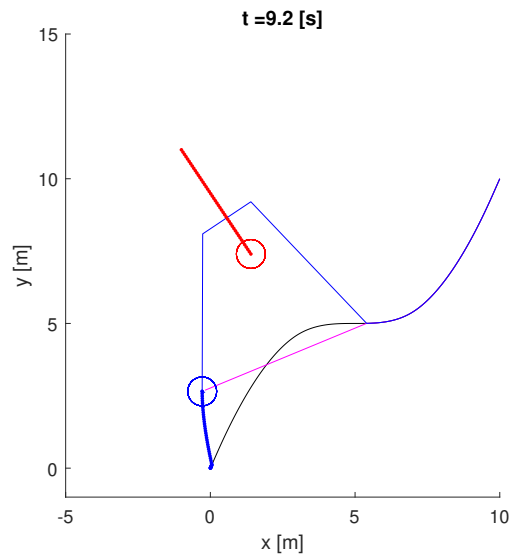


Figure 5.21: Robot continues following the avoidance trajectory

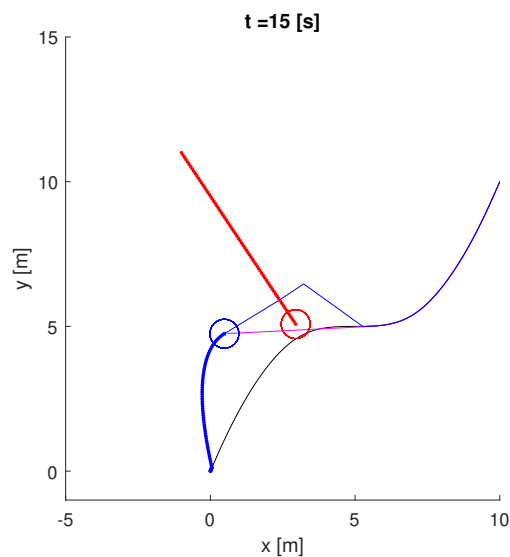


Figure 5.22: The robot will continue to follow the avoidance trajectory until the obstacle is no more in potential collision with the reference one (in magenta)

After that, when the obstacle will be no more in potential collision with the considered reference trajectory, it will merge again in the original reference one (see Figure 5.23) and will continue following it (see Figure 5.24).

5.5 Trajectory modification algorithm

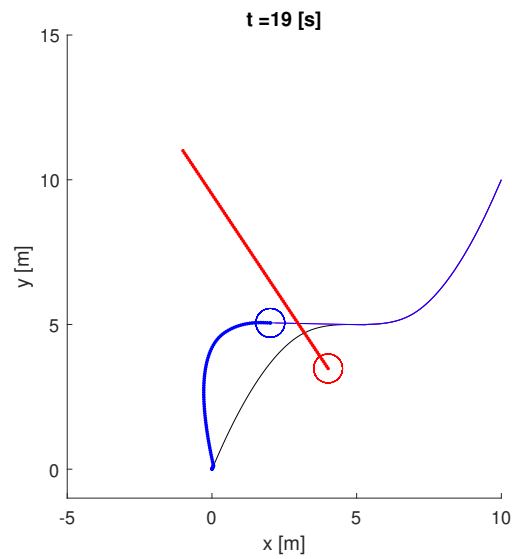


Figure 5.23: After avoiding the obstacle, the robot will continue following the merging trajectory

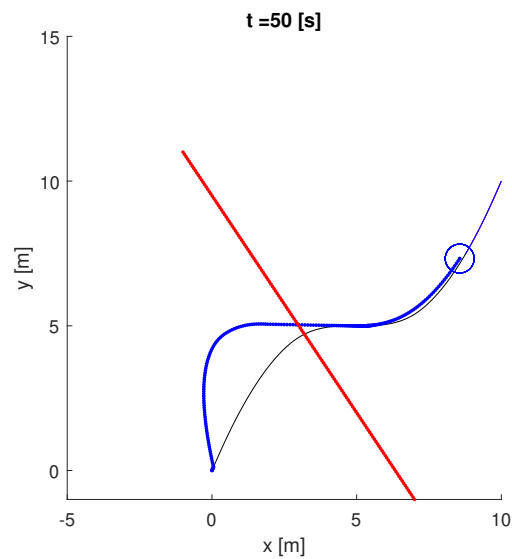


Figure 5.24: The robot finally reaches the original reference trajectory and continues following it

Finally, by analyzing the actual distance kept by the robot from the obstacle, it can be seen that the safety distance has never been violated (see Figure 5.25).

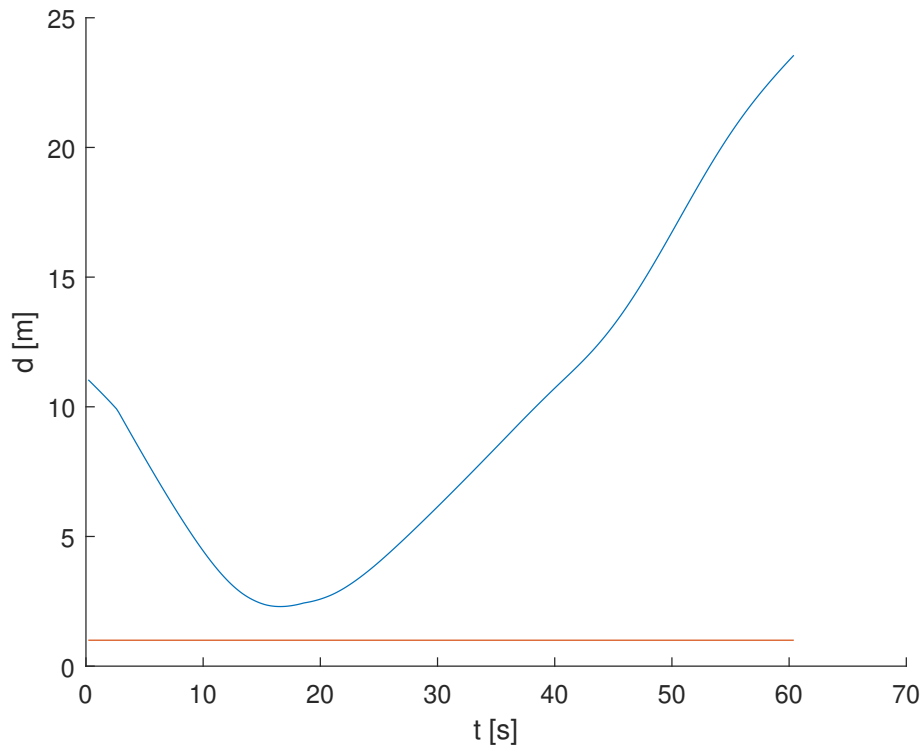


Figure 5.25: Distance between robot and obstacle (on y-axis), with respect to time (on x-axis). The orange line represents the boundary value $d_s = 0.5[m]$ while the blue line represents the actual value during the simulation

5.6 Comparison with position constraints

In order to show the effectiveness of the proposed solution, a situation in which the usage of the position constraints described in section 3.7.1 lead to a collision is compared against the same case using the trajectory modification algorithm described in section 4.2.

In particular, the robot has to follow the reference trajectory shown in Figure 5.26, starting from $P_r = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, while an obstacle starting from

$P_o = \begin{bmatrix} 0 \\ 30 \end{bmatrix}$ is moving with velocity $V_o = \begin{bmatrix} 0.27 \\ -0.76 \end{bmatrix}$.

Additionally, in order emulate the limited range of robot sensors, obstacles are detectable by the robot only within a radius of $10[m]$ from its position.

5.6 Comparison with position constraints

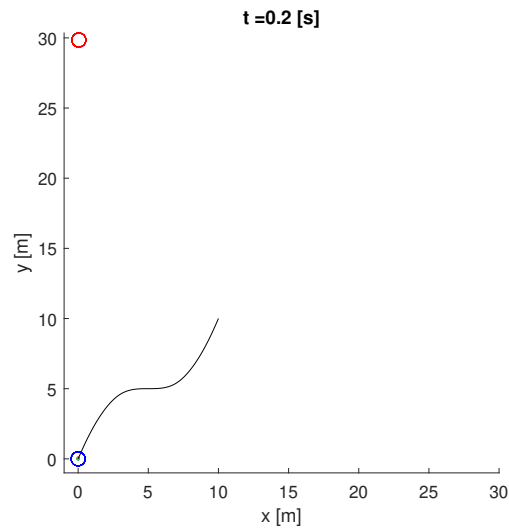


Figure 5.26: Initial situation

The robot follows the trajectory until $t = 20.4[s]$ where the obstacle is detected and position constraints for current and future time instants are set (see Figure 5.27).

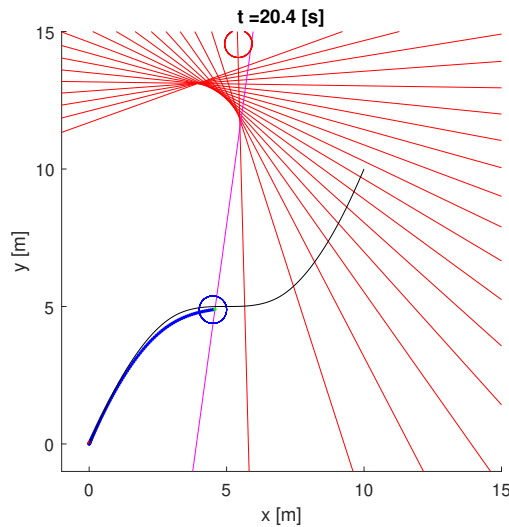


Figure 5.27: Situation in which the robot starts detecting the obstacle. The magenta line represents the ΔV line, while the red ones are the constraints. The red circle represent the obstacle and the blue points represent robot positions

However, due to the higher speed of the obstacle, the robot will continue to follow the reference until reaching the situation in Figure 5.28, where the robot starts violating some of the future position constraints (which,

for simplicity, have been computed by setting a social distance of $0.5[m]$ that takes into account obstacle personal space).

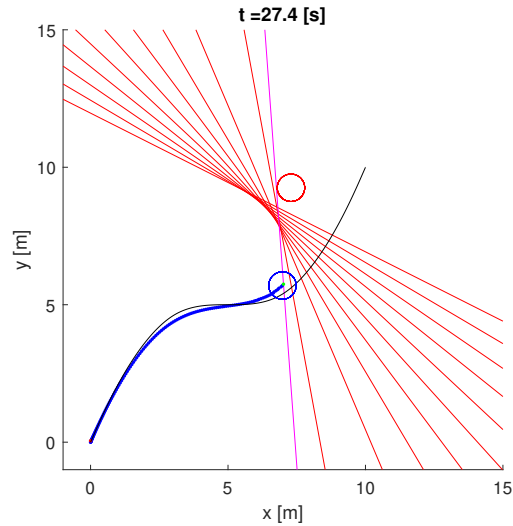


Figure 5.28: The robot continues following the reference trajectory

Finally, due to the limited space and time available to properly avoid the obstacle, the robot and the obstacle will collide, as shown in figure 5.29.

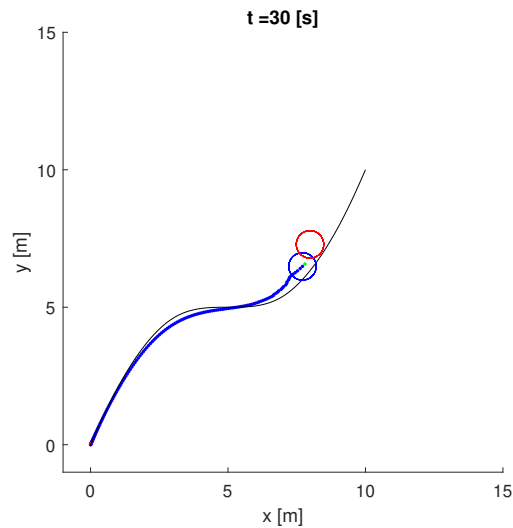


Figure 5.29: Robot is not able to properly avoid the obstacle

By enlarging the colliding area (see Figure 5.30) it can be noticed that before the collision the robot has started increasing its distance from the reference trajectory in order to avoid the obstacle (i.e., the effect of slack variables).

5.6 Comparison with position constraints

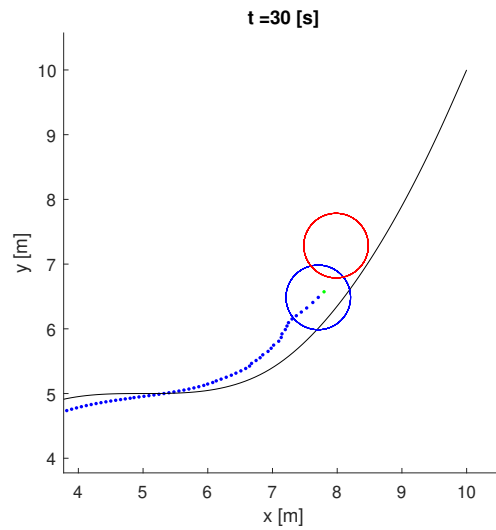


Figure 5.30: Detail of the colliding area

Finally, by plotting the distance d between robot and obstacle in time, it can be seen that at $t = 30[s]$ the safety distance (the orange line) is violated, which means that robot and obstacle were at a distance closer than the sum of their radius.

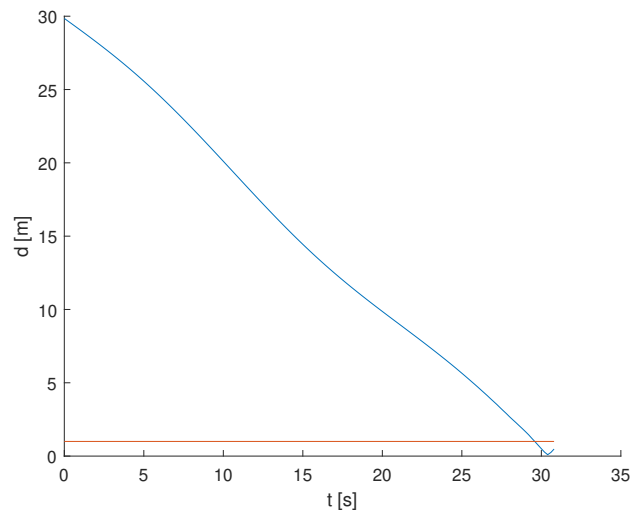


Figure 5.31: Distance between robot and obstacle (on y-axis), with respect to time (on x-axis). The orange line represents the boundary value $d_s = 1[m]$ while the blue line represents the actual value during the simulation

On the other hand, using the algorithm described in section 4.2, when detecting the obstacle at $t = 20.2[s]$, the avoidance trajectory shown in

Figure 5.32 is computed and used as a reference for the MPC reference tracking problem. In contrast, the magenta linear trajectory connects the robot position to the merging point. It will be used as a reference trajectory for the subsequent iterations.

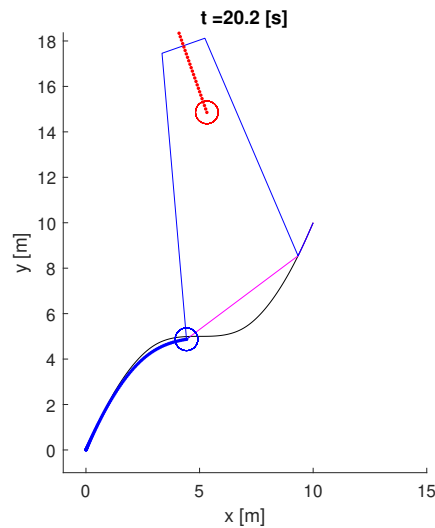


Figure 5.32: The algorithm builds the first avoidance trajectory and modifies the reference trajectory by inserting the magenta merging trajectory

The robot will start following the avoidance trajectory, which will get closer as the obstacle approaches the reference trajectory, as it can be seen from Figure 5.33

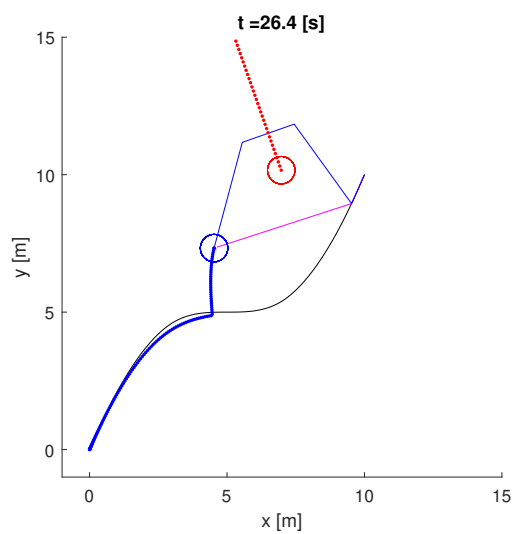


Figure 5.33: Robot follows the avoidance trajectory

5.6 Comparison with position constraints

Then, when the obstacle overcomes the reference trajectory (i.e., the one represented in magenta), the robot will continue following the linear trajectory that will reconnect to the original trajectory in the merging point (see Figures 5.34 and 5.35).

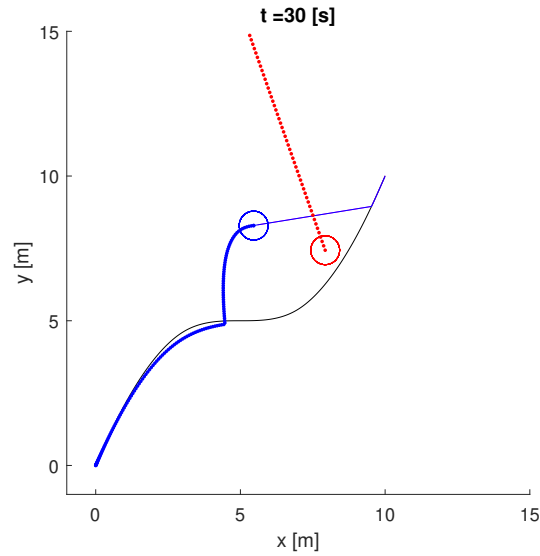


Figure 5.34: After overcoming the obstacle, the robot will follow the merging trajectory

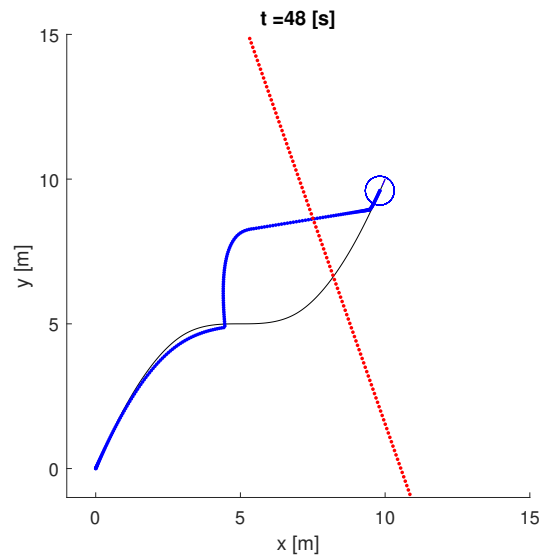


Figure 5.35: After following the merging trajectory, the robot has reached again the original reference trajectory

Finally, by plotting the distance from the obstacle it can be seen that the

robot has successfully kept a distance above $d_s = 0.5[m]$ during all the avoidance procedure.

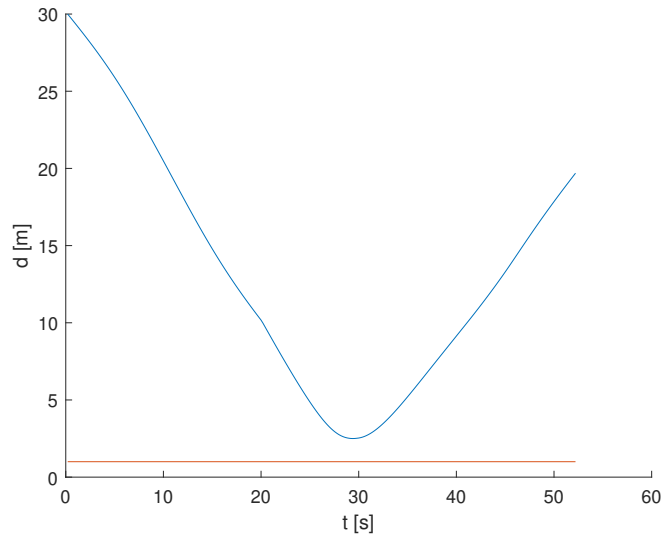


Figure 5.36: Distance between robot and obstacle (on y-axis), with respect to time (on x-axis). The orange line represents the boundary value $d_s = 1[m]$ while the blue line represents the actual value during the simulation

5.7 Avoidance in a low density environment

Recalling the fact that the technique presented in this work refers to low density scenarios, where pedestrians are able to freely modulate their velocity, a situation with three obstacles moving in different directions with different velocities has been set up.

In particular, the reference trajectory is the same used in the previous cases, while the obstacles are characterized by:

$$\begin{aligned}
 P_{obs}^{(1)} &= \begin{bmatrix} -1 \\ 9 \end{bmatrix}, & V_{obs}^{(1)} &= \begin{bmatrix} 0.2 \\ -0.6 \end{bmatrix} \\
 P_{obs}^{(2)} &= \begin{bmatrix} 25 \\ 0 \end{bmatrix}, & V_{obs}^{(2)} &= \begin{bmatrix} -0.6 \\ 0.2 \end{bmatrix} \\
 P_{obs}^{(3)} &= \begin{bmatrix} 20 \\ -10 \end{bmatrix}, & V_{obs}^{(3)} &= \begin{bmatrix} -0.3 \\ 0.4 \end{bmatrix}
 \end{aligned} \tag{5.14}$$

5.7 Avoidance in a low density environment

At the beginning, the robot will detect the first obstacle and build the avoidance trajectory for it (see Figure 5.37).

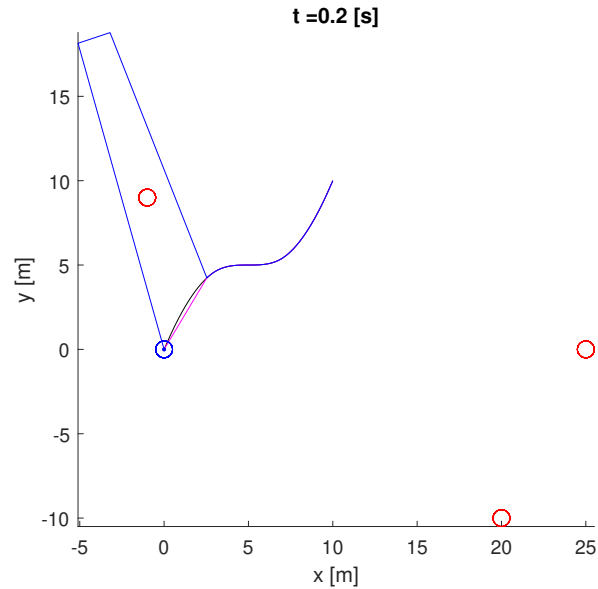


Figure 5.37: Initial situation, the robot starts planning for avoiding the first obstacle

The robot will follow the avoidance trajectory until there is no more potential collision with the obstacle (see Figures 5.38 and 5.39).

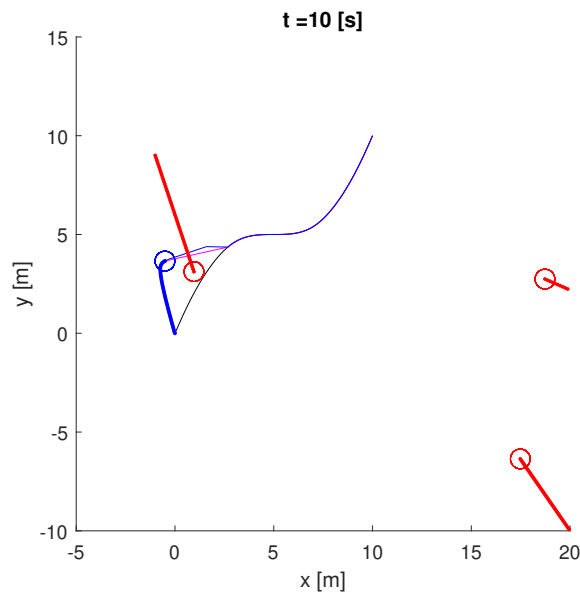


Figure 5.38: Robot is following the avoidance trajectory for the first obstacle

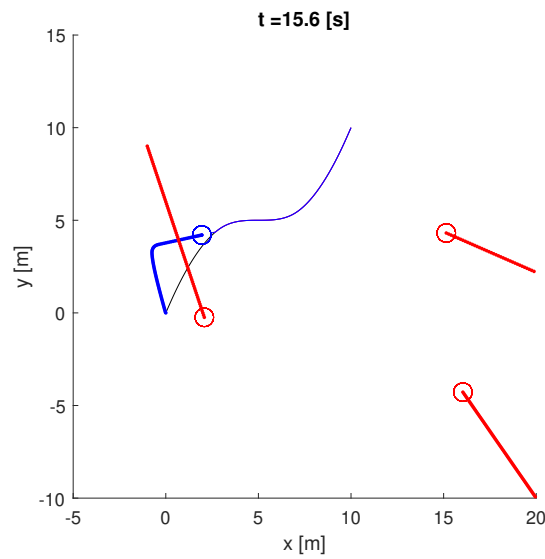


Figure 5.39: The avoidance maneuver for the first obstacle has been successfully completed

Then, the robot will reach again the original reference trajectory before detecting the second obstacle (see Figure 5.40).

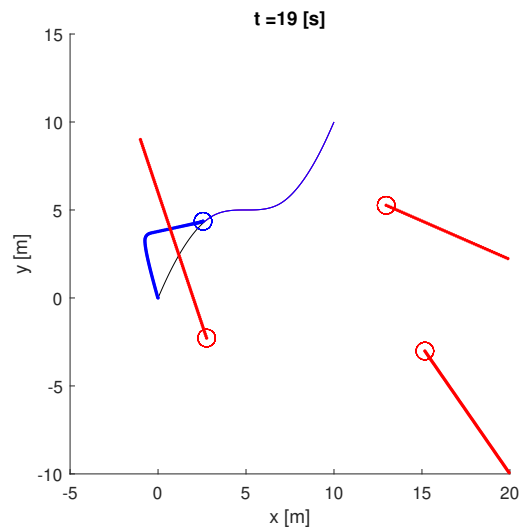


Figure 5.40: Robot has reached again the original reference trajectory

At this point, the robot detects the second obstacle and plans for avoiding it (see Figure 5.41). However, when the robot starts also detecting the third obstacle, it will be seen that it potentially collides with the avoidance trajectory computed for the second one. In that case, the avoidance tra-

5.7 Avoidance in a low density environment

jectory for the third obstacle will be computed with respect to the robot position. Therefore, It will merge in the avoidance trajectory planned for the second obstacle (see Figure 5.42).

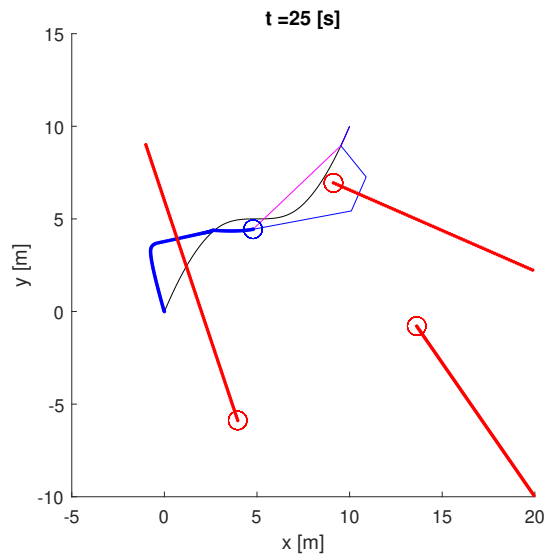


Figure 5.41: Situation in which an obstacle potentially collides with the avoidance trajectory of another obstacle

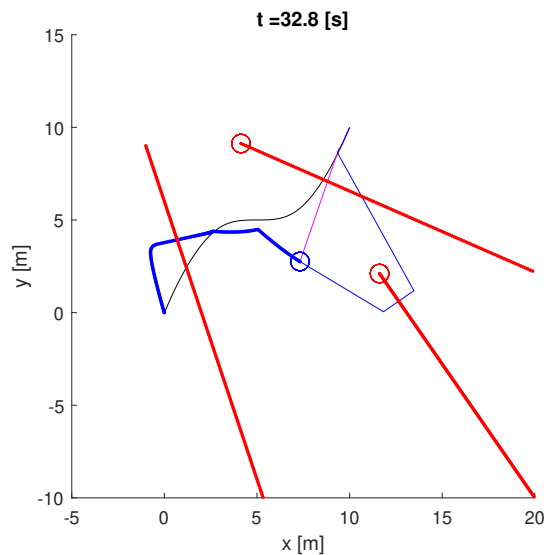


Figure 5.42: After the second obstacle, the robot will continue avoiding the third one

After that, the robot will follow the avoidance trajectory until finally reaching again the original reference trajectory (see Figure 5.43).

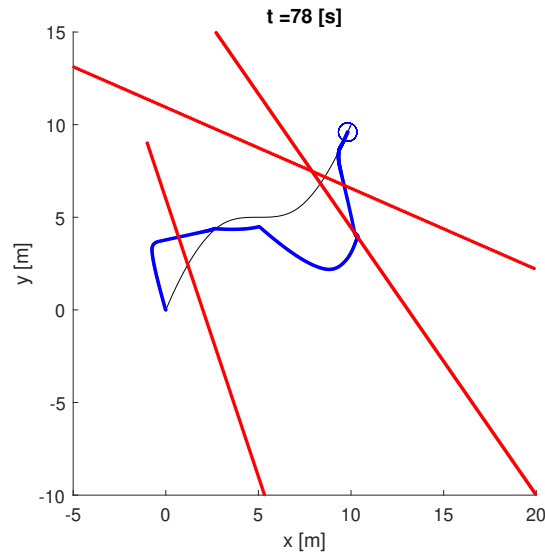


Figure 5.43: Final situation in which the robot has reached again the original reference trajectory

Finally, by plotting the distances from each obstacle in time, it can be seen that the safety distance requirement has been successfully fulfilled for all the obstacles, as it can be seen in Figure 5.44.

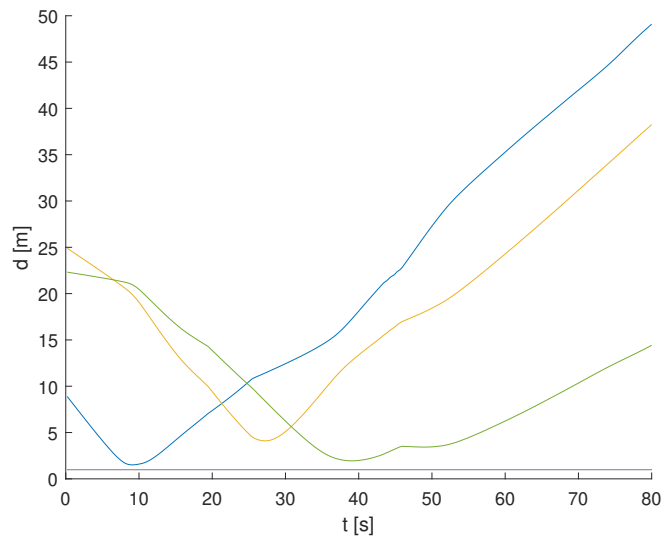


Figure 5.44: Distance between robot and obstacles (on the y-axis), with respect to time (on the x-axis). The purple line represents the boundary value $d_s = 1[m]$ while the blue, yellow and green lines represent the actual value during the simulation

5.7 Avoidance in a low density environment

Another interesting case to be shown is the one in which different obstacles are in potential collision from two opposite sides with respect to the reference trajectory (see Figure 5.45).

In particular, considering two obstacles characterized by:

$$\begin{aligned} P_{obs}^{(1)} &= \begin{bmatrix} 2 \\ 20 \end{bmatrix} & V_{obs}^{(1)} &= \begin{bmatrix} 0.3 \\ -0.8 \end{bmatrix} \\ P_{obs}^{(2)} &= \begin{bmatrix} 25 \\ 0 \end{bmatrix} & V_{obs}^{(2)} &= \begin{bmatrix} -0.6 \\ 0.3 \end{bmatrix} \end{aligned} \tag{5.15}$$

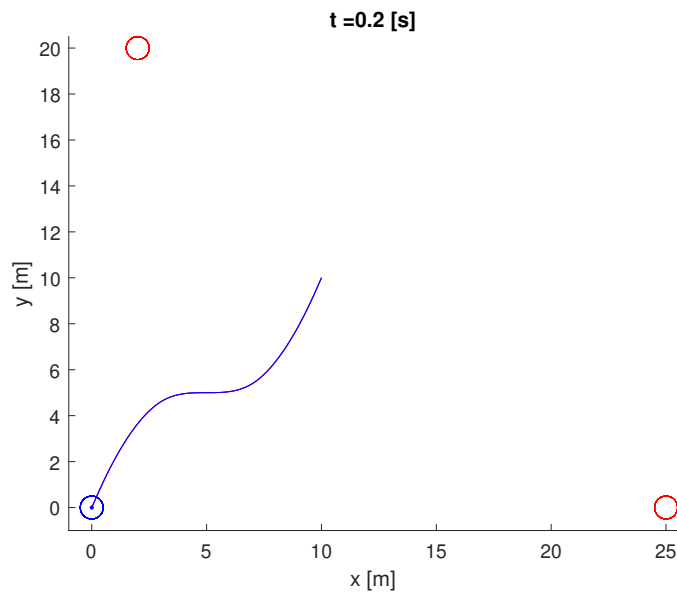


Figure 5.45: Initial situation with two obstacles coming from opposite sides

Therefore, the robot will start following the avoidance trajectory for the first obstacle until detecting the potential collision also with the second one (see Figures 5.46 and 5.47).

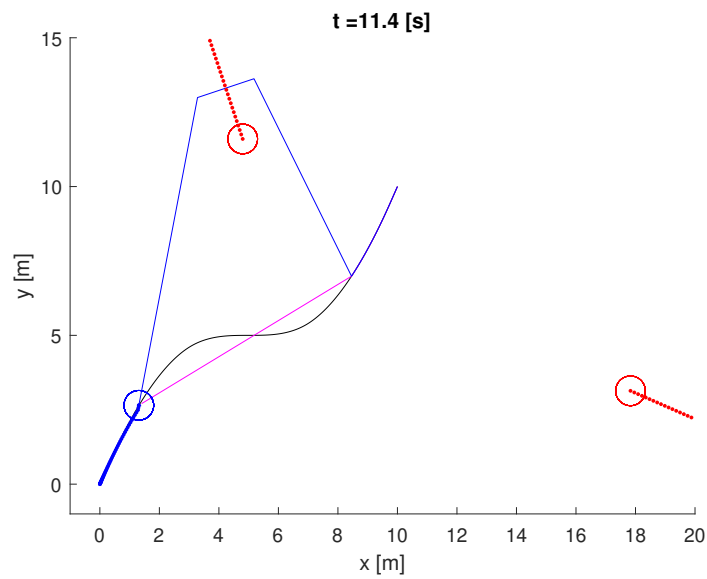


Figure 5.46: The robot plans the avoidance trajectory for the first obstacle

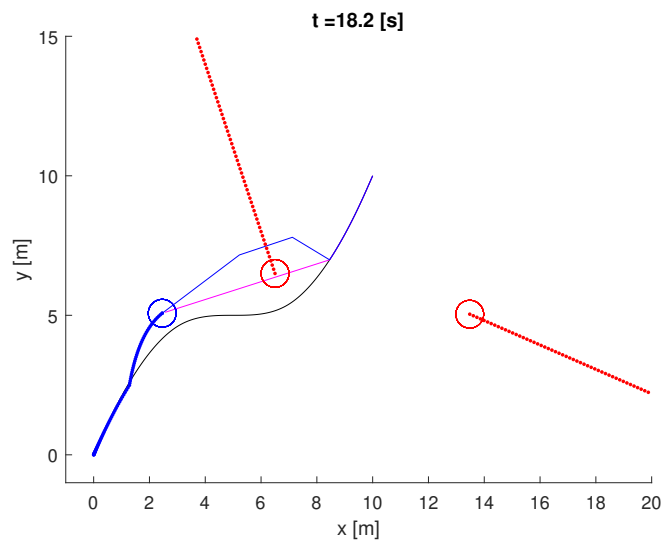


Figure 5.47: The robot continues following the avoidance trajectory for the first obstacle

At this point, the robot should have avoided the first obstacle and planned an avoidance trajectory for the second one. However, the avoidance trajectory with the second obstacle would be in a potential collision with the first one. Thus the avoidance trajectory is modified to take into account the first obstacle trajectory (see Figure 5.48).

5.7 Avoidance in a low density environment

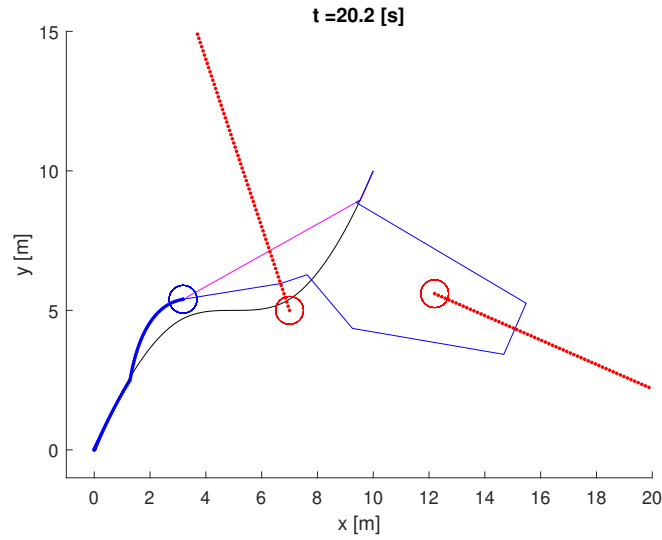


Figure 5.48: The robot has avoided the first obstacle and detects the second one, which has the avoidance trajectory in potential collision with the first obstacle

Furthermore, the robot will follow this avoidance trajectory until only the second obstacle is still in potential collision (see Figure 5.49).

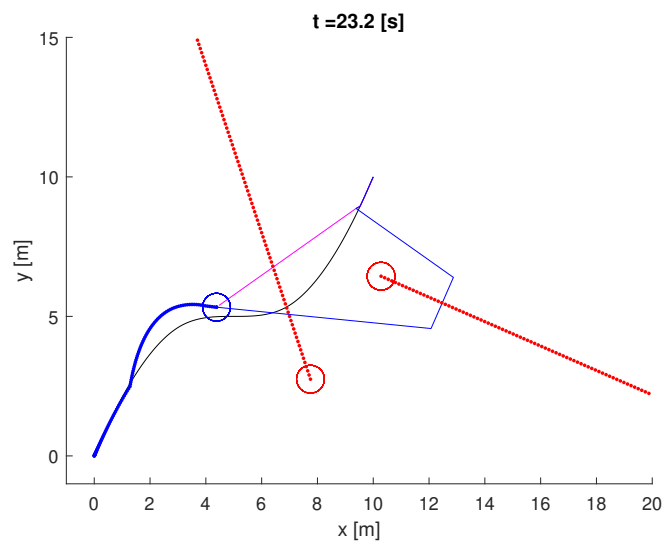


Figure 5.49: The first robot is no more in potential collision with the avoidance trajectory, and the robot continues following it for the second obstacle

Finally, the robot will continue following the modified trajectory until reaching again the original reference (see Figures 5.50 and 5.51).

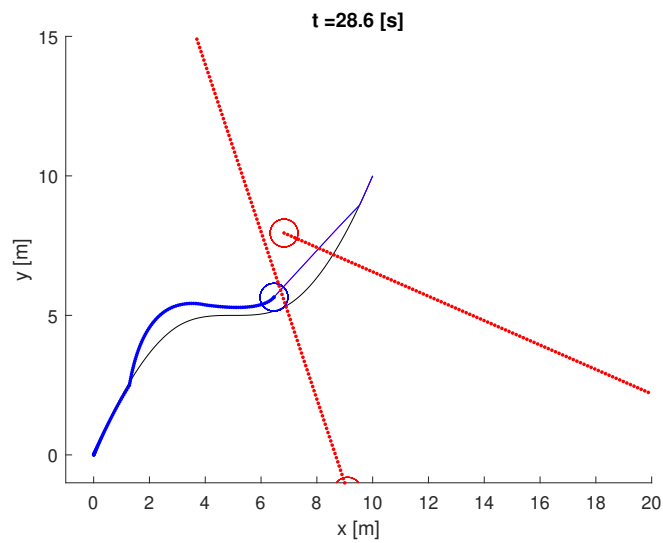


Figure 5.50: The robot has avoided also the second obstacle and follows the merging trajectory

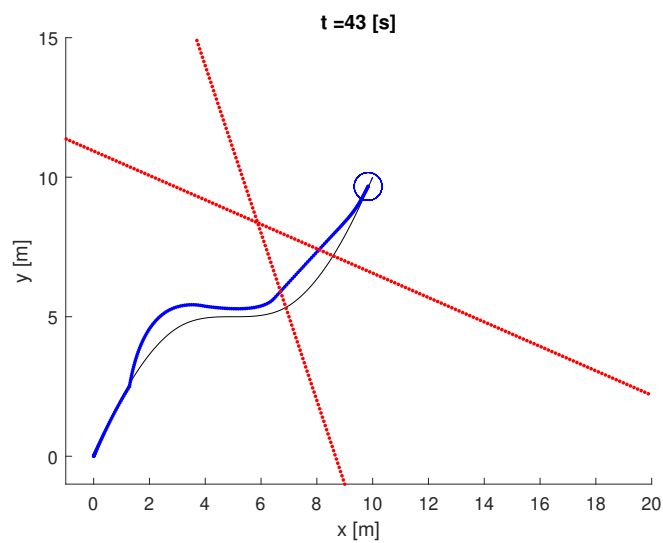


Figure 5.51: The robot has successfully reached again the original reference trajectory

In addition, plotting the distance that the robot has kept from the obstacles, it can be seen that it succeeded in avoiding a collision with both of them.

5.7 Avoidance in a low density environment

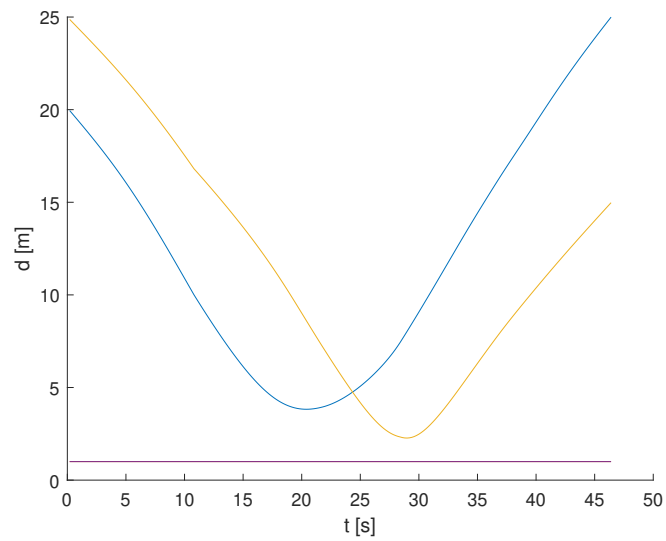


Figure 5.52: Distance between robot and obstacle (on the y-axis), with respect to time (on the x-axis). The purple line represents the boundary value $d_s = 1[m]$ while the blue and yellow lines represent the actual value during the simulation for the first and second obstacle, respectively

Chapter 6

Conclusions and future work

The work developed in this thesis proposes a solution for autonomous navigation in low people density scenarios, where the robot could steer to avoid an obstacle.

In particular, the technique presented relies on the fact that, under the considered assumptions, it could be possible to guide the robot towards a free direction by modifying the trajectory given as a reference to the MPC. In that way, according to the cost function, the controller will choose the optimal solution compatible with the avoidance trajectory, if available.

Simulations have shown that the approach is applicable in different situations and effectively allows the robot to continue its motion while following a locally variable avoidance trajectory, although different observations could be done.

The proposed solution fits well in all the situations where pedestrians move faster than the robot. In that case, the technique presented will allow the robot to follow a trajectory that is coherent with the original reference trajectory, while allowing the obstacles to follow their path. Indeed, in some cases is not possible to define an avoidance trajectory that allows yielding the obstacle, for example, when the robot is on an obstacle trajectory or in a crowded environment, where people tend to move in groups at a slower speed. These situations are the ones in which the usage of position state constraints fits well, as already proven in previous works. Additionally, in the cases in which the robot is close to the obstacle trajectory, the anchor point tends to be far. Thus it could be easy to violate an obstacle in the map, and an avoidance trajectory would not be built. Another particular case is the one in which two obstacles are in potential collision with the robot and between themselves. In that case, only one of the two avoid-

ance trajectories will be taken by the algorithm. Although in a realistic situation, one of the two obstacles will have to change either its velocity or trajectory to avoid the collision with the other one and that behaviour is unpredictable.

Therefore, due to the recent developments in computer vision and artificial intelligence, robot perception could be extended by equipping it with a 3D camera to detect obstacle shapes better and eventually predict their motion. This extension would require introducing the new coordinate both at the Global Planning and Local Planning levels.

Additionally, in a smart building context, cameras across the building could be exploited to obtain a real-time probabilistic map reflecting the presence of groups of people, bringing that information at the Global Planner level.

Bibliography

- [1] J. Borenstein and Yoram Koren. Histogramic in-motion mapping for mobile robot obstacle avoidance. *Robotics and Automation, IEEE Transactions on*, 7:535 – 539, 09 1991.
- [2] M. A. Abbas, R. Milman, and J. M. Eklund. Obstacle avoidance in real time with nonlinear model predictive control of autonomous vehicles. In *2014 IEEE 27th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–6, 2014.
- [3] C. Wang, A. S. Matveev, A. V. Savkin, T. N. Nguyen, and H. T. Nguyen. A collision avoidance strategy for safe autonomous navigation of an intelligent electric-powered wheelchair in dynamic uncertain environments with moving obstacles. In *2013 European Control Conference (ECC)*, pages 4382–4387, 2013.
- [4] Hamid Teimoori and Andrey V. Savkin. A biologically inspired method for robot navigation in a cluttered environment. *Robotica*, 28(5):637648, 2010.
- [5] Erwin Prassler, Jens Scholz, and Paolo Fiorini. A robotic wheelchair for crowded public environments. *IEEE Robotics and Automation Magazine*, 8:38–44, 04 2001.
- [6] Takashi Chikayama and Franck FEURTEY. Simulating the collision avoidance behavior of pedestrians. 01 2021.
- [7] Silvia Tellarini. Autonomous navigation in human crowded environments: integrating model predictive control within global planning. Master’s thesis, Politecnico di Milano, 4 2020.
- [8] Mauro Gabellone Eugenio Ceravolo. Controllo del moto di una sedia a rotelle autonoma mediante tecniche di controllo predittivo. Master’s thesis, Politecnico di Milano, 9 2016.

- [9] Md Anowar Hossain, Md. Fazlul Karim Khondakar, Md Sarowar, and Md Qureshi. Design and implementation of an autonomous wheelchair. pages 1–5, 12 2019.
- [10] Dr. Emna Baklouti, Prof. Nader Ben Amor, and Prof. Mohammed Jallouli. Autonomous wheelchair navigation with real time obstacle detection using 3d sensor. *Automatika*, 57(3):761–773, 2016.
- [11] Ugo Rosolia, Stijn Bruyne, and A.G. Alleyne. Autonomous vehicle control: A nonconvex approach for obstacle avoidance. *IEEE Transactions on Control Systems Technology*, 25:1–16, 06 2016.
- [12] Hoang Trieu, Hung Nguyen, and Keith Willey. Advanced obstacle avoidance for a laser based wheelchair using optimised bayesian neural networks. *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*, 2008:3463–6, 02 2008.
- [13] Gianluca Bardaro, Luca Bascetta, Eugenio Ceravolo, Marcello Farina, Mauro Gabellone, and Matteo Matteucci. Mpc-based control architecture of an autonomous wheelchair for indoor environments. *Control Engineering Practice*, 78:160–174, 09 2018.
- [14] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. volume 3, 01 2009.
- [15] move_base official documentation. http://wiki.ros.org/move_base.
- [16] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of a*. *J. ACM*, 32(3):505–536, July 1985.
- [17] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.
- [18] A. Kormanová, M. Varga, and N. Adamko. Hybrid model for pedestrian movement simulation. In *The 10th International Conference on Digital Technologies 2014*, pages 152–158, 2014.
- [19] Andreas Gustavsson. An efficient approach for detecting moving objects and deriving their positions and velocities. In Kohei Arai and

BIBLIOGRAPHY

- Supriya Kapoor, editors, *Advances in Computer Vision*, pages 293–313, Cham, 2020. Springer International Publishing.
- [20] Lalo Magni and Riccardo Scattolini. *Advanced and multivariable control*. Pitagora, 2014.
- [21] G. Oriolo, A. De Luca, and M. Vendittelli. Wmr control via dynamic feedback linearization: design, implementation, and experimental validation. *IEEE Transactions on Control Systems Technology*, 10(6):835–852, 2002.
- [22] Weihua Chen, Tie Zhang, and Yanbiao Zou. Mobile robot path planning based on social interaction space in social environment. *International Journal of Advanced Robotic Systems*, 15(3):1729881418776183, 2018.
- [23] Misiano S. Studio di tecniche distribuite di controllo predittivo stocastico per inseguimento di segnali di riferimento e applicazione al coordinamento di robot mobili. Master’s thesis, Politecnico di Milano, 2014.
- [24] Chien-Yen Chang, Hugh Woo, Sen-Feng Wang, and Ming-Jen Kuo. Survey and analysis of walking speeds for the elderly and children at crosswalks. 25, 06 2010.
- [25] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17:760–, 07 1998.

