

**POLITECNICO DI MILANO**  
Corso di Laurea Magistrale in Ingegneria dell'Automazione  
Dipartimento di Elettronica e Informazione e Bioingegneria



# **HUMAN-AWARE PLANNING AND CONTROL OF AN INDOOR AUTONOMOUS WHEELCHAIR**

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

**Tesi di Laurea di:**  
**Pierpaolo Durante, matricola 863104**

**Anno Accademico 2017-2018**



*Ai miei nonni: Evelina, Gina, Gabriele e Pietro*



# Abstract

The work done in this thesis focuses on the motion planning of an AGV (Autonomous Guided Vehicle), in particular an indoor autonomous wheelchair. The main goal is to develop a solution in order to make the vehicle completely human-aware and able to move inside a crowded environment. The adopted strategy relies on two different levels of control: *Global Planning*, on one side, and *Local Planning*, on the other.

In particular, the global planner focuses on the computation of an optimal trajectory, through a novel application of the approach called *RRT\* Motion Primitives*, by taking into account the characterization of human crowds, represented by means of a probabilistic distribution. The computed probabilities are used to construct a probabilistic map in which the trajectory planning will be performed. The optimal trajectory is then segmented in order to define a sequence of desired positions for the wheelchair.

Aim of the local planner is to control the vehicle position, by solving a regulation problem and by imposing to the system the input values of longitudinal and angular velocity of the wheelchair, in order to reach each point of the segmented trajectory, guaranteeing collision avoidance with fixed and moving obstacles, comfort for the passenger and safety for the pedestrians. The local planning solution is obtained through the development of an advanced control strategy called *MPC (Model Predictive Control)*, which allows to directly introduce in the formulation the limitations related to the system, by means of constraints of an optimization problem.

The results observed from simulations proved the effectiveness of the combination of the two strategies.



# Sommario

Il lavoro svolto in questa tesi si colloca nell'ambito della pianificazione del moto di un AGV (Autonomous Guided Vehicle), in particolare una carrozzina autonoma. L'obiettivo principale è quello di sviluppare una soluzione che renda il veicolo completamente conscio della presenza umana e capace di muoversi all'interno di un contesto caratterizzato da folle di persone. La strategia adottata si basa su due livelli di controllo: da un lato la *Pianificazione Globale*, dall'altro la *Pianificazione Locale*.

In particolare, il pianificatore globale si focalizza sul calcolo di una traiettoria ottima, attraverso una innovativa applicazione dell'approccio denominato *RRT\* Motion Primitives*, il quale prende in considerazione folle di persone, caratterizzandole attraverso una distribuzione probabilistica. Le probabilità calcolate sono utilizzate per costruire una mappa probabilistica nella quale la pianificazione della traiettoria verrà eseguita. La traiettoria ottima è in seguito segmentata con lo scopo di definire posizioni strategiche che dovranno essere raggiunte dal veicolo.

L'obiettivo del pianificatore locale è di controllare la posizione della carrozzina, risolvendo un problema di regolazione e fornendo in ingresso al sistema i valori di velocità longitudinale ed angolare del veicolo, per raggiungere ogni punto della traiettoria segmentata, evitando ostacoli fissi e mobili e garantendo contemporaneamente i requisiti di sicurezza e comfort per il passeggero e per i pedoni. La soluzione legata alla pianificazione locale è ottenuta attraverso lo sviluppo di una tecnica di controllo avanzata, denominata *MPC (Model Predictive Control)*, la quale permette di introdurre direttamente nella formulazione i limiti legati al sistema, attraverso la definizione di vincoli all'interno di un problema di ottimizzazione. I risultati ottenuti per mezzo di simulazioni hanno dimostrato l'efficacia della combinazione tra le due strategie.



# Ringraziamenti

Prima di tutto vorrei ringraziare il mio relatore, il Prof. Luca Bascetta, per la sua immensa disponibilità, per i suoi attenti consigli e per avermi dato la possibilità di apprendere molto e di lavorare ad un progetto a cui tenevo particolarmente.

Vorrei inoltre ringraziare la Prof. Maria Prandini per la sua gentilezza e per i suoi indispensabili suggerimenti, grazie ai quali questa tesi ha potuto prendere forma.

Vorrei esprimere la mia più sincera gratitudine a Basak Sakcak, una brillante e generosa ricercatrice che non ha esitato a mettermi a disposizione il suo tempo ed il suo prezioso progetto, rendendo unico questo lavoro.

Ringrazio i miei genitori per il loro affetto e per aver sempre creduto in me.

Infine dedico un ringraziamento speciale alla mia famiglia e a tutte le persone a me vicine che hanno contribuito a rendere meno tortuoso questo percorso.



# Indice

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Aim of the thesis . . . . .	3
1.2	Thesis Structure . . . . .	4
<b>2</b>	<b>Planning and Control</b>	<b>7</b>
2.1	Global Planning . . . . .	9
2.1.1	State of the Art . . . . .	9
2.1.2	Planning Solution . . . . .	11
2.2	Local Planning . . . . .	11
2.2.1	State of the Art . . . . .	12
2.2.2	Control Solution . . . . .	13
<b>3</b>	<b>Robot and Human Models</b>	<b>15</b>
3.1	Vehicle . . . . .	15
3.1.1	Unicycle Model . . . . .	16
3.1.2	Feedback Linearization . . . . .	18
3.2	Pedestrian . . . . .	20
3.2.1	Kinematic of the Model . . . . .	21
3.2.2	Virtual Box . . . . .	21
3.3	Walls and Fixed Obstacles . . . . .	24
3.4	Crowd . . . . .	25
<b>4</b>	<b>Model Predictive Control</b>	<b>29</b>
4.1	MPC Formulation . . . . .	29
4.1.1	MPC with Receding Horizon . . . . .	30
4.1.2	Constraints . . . . .	30
4.1.3	Stability Analysis . . . . .	32
4.2	Vehicle Model for Control . . . . .	33
4.3	Cost Function . . . . .	34
4.3.1	Controller Tuning . . . . .	35
4.4	Constraint Definition . . . . .	37
4.4.1	Position Constraints . . . . .	37
4.4.2	Velocity Constraint . . . . .	56

---

4.4.3	Velocity Variation Constraints . . . . .	59
4.5	Slack Variables . . . . .	60
4.5.1	Cost Function . . . . .	61
4.5.2	Soft Position Constraints . . . . .	62
4.5.3	Soft Velocity Variation Constraints . . . . .	65
4.5.4	Slack Variables Constraints . . . . .	66
4.6	Final Pose Control . . . . .	67
<b>5</b>	<b>RRT* Motion Primitives</b>	<b>73</b>
5.1	Motion Planning Problem . . . . .	73
5.1.1	Kinodynamic Motion Planning . . . . .	75
5.2	RRT . . . . .	76
5.3	RRT* . . . . .	79
5.4	RRT* Motion Primitives . . . . .	81
5.4.1	Sampling-Based planning with Motion Primitives . . . . .	81
5.4.2	Motion Planning . . . . .	83
5.5	RRT* MP on Probabilistic Maps . . . . .	89
<b>6</b>	<b>Simulations Results</b>	<b>95</b>
6.1	Local Planner . . . . .	95
6.1.1	Convergence to the Goal . . . . .	97
6.1.2	Consistency of Velocity and Velocity Variation Constraints . . . . .	98
6.1.3	Pedestrian Avoidance . . . . .	99
6.1.4	Consistency of Constraints with Slack Variables . . . . .	102
6.1.5	Complete Simulations . . . . .	103
6.2	Global Planner . . . . .	106
<b>7</b>	<b>Conclusions</b>	<b>115</b>
	<b>Bibliography</b>	<b>119</b>

# Capitolo 1

## Introduction

The work done in this thesis focuses on the motion planning of autonomous vehicles, known in the literature as AGVs (Autonomous Guided Vehicles). The use of this particular kind of portable robots broadened in the industrial field during the ending of the 20<sup>th</sup> century. However, in the last few years, the application of AGVs was extended to different and more particular fields, as Automotive, Manufacturing, Healthcare and even Theme Parks.

The large versatility of AGVs is related to the fact that these vehicles can solve many tasks in autonomy by reducing, or even removing, the human intervention. The main problems in designing AGVs are mostly related to autonomous navigation, i.e. trajectory generation and obstacle detection and avoidance inside the environment in which the robot will perform its motion.

### 1.1 Aim of the thesis

The thesis focuses on the research in the field of Human-Robot Interaction (HRI), in particular on the navigation of a given automated vehicle in a context in which the human presence plays a fundamental role. The analysis takes into account a crowded environment, in which the robot will navigate with the purpose of reaching a desired location on a map and, at the same time, performing collision avoidance in order to guarantee safety for humans.

In particular, this case study focuses on the navigation system of an autonomous wheelchair, in order to assist patients in hospitals and, in general, people with motor disability. Furthermore there are some diseases, like the ALS (Amyotrophic Lateral Sclerosis) or the Parkinson, which compromise the ability of a patient to autonomously control the movement of an electric wheelchair and, for this reason, the adoption of an AGV turns out to be a good solution. A previous thesis puts the basis for this case study. In particular, in the cited document, the realization of a navigation system is described, focusing on the adoption of an advanced control strategy, called

*Model Predictive Control*. The effectiveness of this approach in the context of local planning and with respect to fixed obstacles has been successfully validated through the development an experimental set-up. In addition, a second work, presenting a novel strategy in the field of trajectory planning, called RRT\* Motion Primitives, is taken into account.

Aim of this thesis is to combine the two approaches in a novel way, in particular by developing a human-aware planning and control strategy. The RRT\* Motion Primitives will be used to compute a human-aware optimal trajectory inside a crowded context. The trajectory is then segmented and, for each segment a regulation problem is solved, in order to bring the vehicle to each desired location, trough the implementation of the Model Predictive Control strategy. To this aim, the solution proposed in the previous work will be improved in order to take into account the presence of moving obstacles, as pedestrians. The results obtained from simulations show the effectiveness of the introduced improvements and the completeness of the navigation system, achieved with the coordination between the two approaches.

## 1.2 Thesis Structure

In the following, the thesis structure will be presented:

- Chapter 2 introduces the problem related to the human-aware robot navigation, focusing on the distinction between *Local* and *Global Planning*. After the introduction of the state of the art, the solution formulated in this thesis, based on the adoption of *Model Predictive Control* for Local Planning and *RRT\* Motion Primitives* for Global Planning, will be highlighted.
- Chapter 3 presents the mathematical models describing the vehicle and a single pedestrian, necessary to develop the Local Planner. The second part focuses on the characterization of a crowded environment in order to construct probabilistic maps, required for the approach adopted in Global Planning.
- In Chapter 4, the Model Predictive Control approach considered in this work will be introduced. In the first part, the canonical formulation of the control problem will be presented in order to put the basis for the solution adopted in this work. Particular attention will be given to the main advantages of using this strategy in the context of Local Planning.
- Chapter 5 will focus on the Global Planning and, in particular, in the description of the RRT (Rapidly-Exploring Random Tree) approach and its variants, focusing on the RRT\* Motion Primitives strategy,

adopted in this context. In the last part, the method to implement such algorithm inside an environment characterized by the human presence, through the introduction of probabilistic maps, will be described.

- Chapter 6 will show all the results obtained through simulations, justifying the implementation choices, related to the control parameters. Particular attention will be given to the system behaviour in different scenarios.
- In Chapter 7, the conclusions retrieved from the results obtained will be reported and a short discussion of possible future developments will be presented, focusing on future improvements.



## Chapter 2

# Planning and Control

A wheelchair can be considered autonomous if it can independently perform a series of tasks without the intervention of the patient or the operators. In particular, it should be designed to reach a desired location by ensuring the comfort of the passenger. To this aim, it is important to impose to the vehicle a limit in velocity, in order to make the passenger feeling safe, but also in its variation. The last aspect plays a fundamental role in order to avoid wheels slipping and also because a sudden change in acceleration can be felt by a patient as an unsafe behaviour.

In the field of trajectory planning, the vehicle should be aware that an indoor context is characterized by a series of fixed obstacles (as walls and furnitures), but also by moving obstacles, in particular humans. This aspect is not so trivial, because in general AGVs, in particular in the industrial application field, can more easily detect and predict the motion of moving obstacles, which can be other robots, designed in order to collaborate with other individuals, or humans, which in general are specialized workers, trained to rationally react to particular circumstances. On the contrary, there are contexts in which humans are not used to coexist and cooperate with autonomous vehicles and are not able to safely behave in that situation. By taking into account a crowded indoor situation, as an hospital, or an outdoor environment, there is a high probability to find heterogeneous groups of people, partially composed of elders and children, and therefore it is not possible to assume that a collaborative external behaviour will take place.

The last aspect to be analysed is the fact that common environments are characterized by a series of social rules as, for example, the formation of crowds. In this context, the robot should be able to predict the formation of a group of people and, at the same time, avoid to generate a trajectory too much close to high human-density areas. On the other side, it may happen that a single individual will separate from the formation, so the AGV should be prepared to reactively detect the single pedestrian, identifying it

as a moving obstacle.

In [24] a survey with a large variety of approaches related to the navigation of an autonomous vehicle inside a context characterized by humans, also known as *Human-aware Robot Navigation*, is presented.

The most complete solution, adopted in this thesis, is based on the introduction of a method formulated on two levels, as shown in Figure 2.1, in order to guarantee both Planning and Control. The layer specialized in planning, also known as *Global Planner*, will compute an optimal trajectory, considering fixed obstacles and the presence of crowds. The so obtained trajectory will be then segmented into a series of points. For every segment, the controller, also known as *Local Planner*, will solve a regulation problem to reach each point (seen as goals) and, at the same time, will reactively act in order to detect obstacles and ensure a collision avoidance.

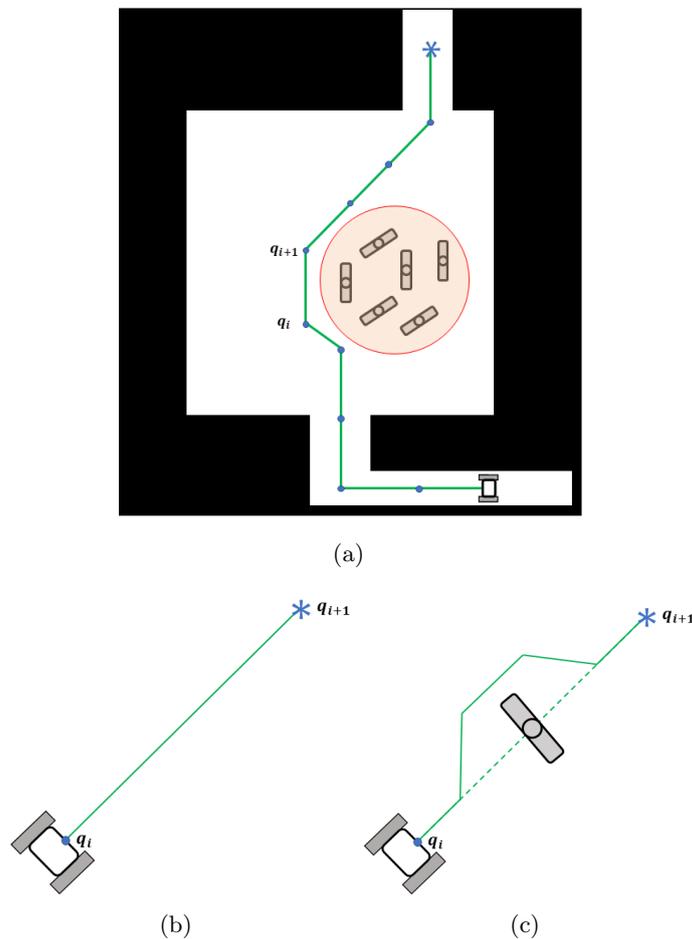


Figure 2.1: Planning and Control: (a) Global Planning in a Crowded Environment (b) Local Planning Regulation (c) Local Planning Regulation in Presence of Pedestrians

## 2.1 Global Planning

Path planning, also known in the robotic literature as Global Planning, is a field of study whose purpose is to compute a desired movement for a robot while satisfying motion constraints and guaranteeing optimality. The most simple problem is based on producing a continuous path in order to connect a starting configuration with a goal and, at the same time, avoiding collision with obstacles, as shown in Figure 2.2.

Plenty of algorithms are available in the robotic literature to solve the path planning problem, but each one of them is based on the notion of *Configuration Space*. To this aim, by defining a single configuration  $q(t) \in \mathbb{R}^n$  as the vector of generalized coordinates describing the robot position inside a n-dimensional space, the Configuration Space  $Q$  is the set of all the different configurations. Moreover, it is possible to define the set of the collision free configurations as:

$$Q_{free} = Q \setminus Q_{occ}$$

where  $Q_{occ}$  is the set of configurations that may cause collision with the obstacles inside the environment.

The above definitions allow to introduce the state of the art in the field of global planning.

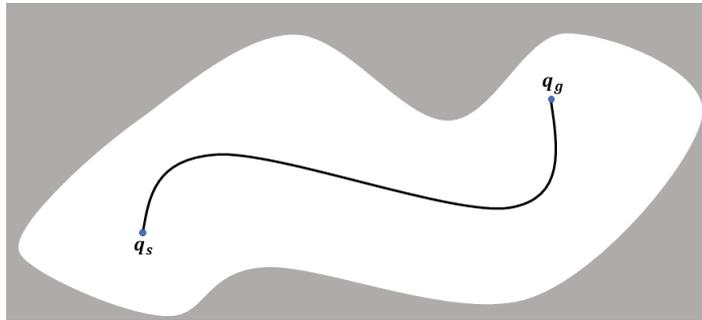


Figure 2.2: Example of a Path Planning Problem from  $q_s$  to  $q_g$ . In white the Collision Free Configuration Space  $C_{free}$ , in grey the Occupied Configuration Space  $C_{obs}$ .

### 2.1.1 State of the Art

In [6] different planning algorithms are presented.

#### Cell Decomposition

This family of algorithms, presented for the first time in [12], are based on the decomposition of the collision-free configuration space  $Q_{free}$  into geometrical shapes called *cells*. Given a starting configuration  $q_s$  and a goal configuration

$q_g$  it is clear that, if they belong to the same cell or to two adjacent cells, a collision free path connecting them can be easily computed. Different motion planning methods based on this approach have been presented in the past years, by distinguishing different kind of decompositions related on the type of cell. However, the main idea is based on the construction of a *connectivity graph* in which the nodes coincide with the cells, while the arcs represent the fact that two cells are adjacent.

### Artificial Potential Fields

These approaches ([16],[10]) approximate the robot as a particle moving inside a potential field, obtained through the definition of an attractive potential near the goal and a repulsive potential near the obstacles. The main drawback of this strategy is related to the fact that an optimal solution cannot be always achieved, due to the presence of local minima.

### Search Based

This class of algorithms is based on the superposition of a grid over the Configuration Space. Each point of the grid identifies a single configuration of the robot, which is allowed to move towards adjacent grid points. As highlighted in [2], the path connecting two points can be computed through the definition of feasible trajectories, called *Motion Primitives*, and it is extracted through the use of search algorithms, as  $A^*$  or  $D^*$ , whence the name of the approach. However, the performance of these particular algorithms is highly affected by the value of the grid resolution, which affects the optimality of the solution on one side, but also the execution time on the other.

### Sampling Based

Sampling-Based algorithms ([7],[21],[5]) rely on the selection of sample states from the Configuration Space. At each iteration of the algorithm a sample is randomly selected from the nearby configurations, with respect to the vehicle position, and it is checked if the path connecting two samples is collision free. If the condition is satisfied, the sample will be added to a tree, whose nodes identify the robot configurations, while the arcs coincide with the collision free paths connecting two adjacent nodes and computed on-line, through a procedure called *steering*. The final result is the extraction of a roadmap from the graph connecting a starting configuration  $q_s$  with a goal  $q_g$ . The most important examples of this kind of approaches are the PRM (Probabilistic Roadmap [13]) and the RRT (Rapidly-exploring Random Tree [22]), which belong to a subclass of the sampling-based algorithms called *Probabilistic Methods*.

### 2.1.2 Planning Solution

The solution presented in this work relies on the algorithm described in [5]. This approach, which takes the name of *RRT\* Motion Primitives*, relies on the combination of a sampling-based approach and of a search-based one. In particular, the  $RRT^*$  method will be used to construct a tree in which the various robot configurations are connected through the definition of paths. However, in order to reduce the computational weight of the steering procedure, used to connect the configurations in  $RRT^*$ , the paths are not computed on-line, but instead they are extracted from a database of motion primitives, retrieved through a search-based procedure.

The original contribution presented in this thesis is related to the fact that this particular method can be introduced not only for a context characterized by fixed obstacles, but also inside a crowded environment. In particular, the trajectory obtained from the  $RRT^*$  Motion Primitives approach will be not only *Collision-Free*, but also *Human-aware*. In fact, the optimal trajectory will prefer low populated areas, with respect to crowded ones. To this aim, the robot will be characterized by a priori knowledge of the environment, described through the representation of the obstacles from the plan of a building and the representation of the crowds through the use of a probabilistic map, in which each crowd is characterized by a uniform probability distribution, constructed with the information retrieved from external devices, as surveillance cameras.

The obtained result will be then used by a second level of control, the Local Planning, whose aim is related on reaching the discrete points of the segmented optimal trajectory while avoiding collision with each single pedestrian inside the nearby area around the vehicle position.

## 2.2 Local Planning

Local Planning can be seen as a control strategy in which, by knowing the model describing the system and a desired output, it is possible to design a suitable input, acting in the immediate future.

In the field of robotics and, in particular in the context of *Human-aware Navigation*, the main feature of local planners is related to the fact that a robot should perform its motion while ensuring safety and comfort for the humans inside the environment. To this aim, a fundamental property is related to the ability of the robot to guarantee collision avoidance, with respect to fixed and moving obstacles.

In [24] a large variety of algorithms introduced in the field of Human-aware Navigation are presented, by highlighting the main characteristics of a local planning navigation framework inside a crowded environment. In particular:

- the robot shall be aware of its own dynamics, described by a particular model which shall take into account the limitations related to the system;
- the future dynamics of the moving obstacle inside the environment shall be taken into account and, if the time evolution of the obstacle motion is not known, the robot shall be able to predict it with an acceptable degree of approximation;
- the knowledge of the surrounding environment plays an important role. In particular, the robot will need updated information in order to reactively behave to sudden changes. For this reason, human-aware local planners rely in general on the data received from external devices, such as laser sensors and cameras;

With the above conditions, it is possible to introduce the state of the art in the context of human-aware local planning.

### 2.2.1 State of the Art

In [24], a large versatility of approaches related to human-aware local planning are described. In this particular context, a key role is played by the concept of *Collision Avoidance*. A given agent can ensure this property if, for every fixed or moving obstacle, it is possible to generate a collision-free path, while maintaining a safety distance. As already introduced, the property can be achieved only if a given robot is aware of its own dynamics and the behaviour of each individual inside the environment. For this reason, the field of robotics related to human-aware navigation is characterized by a large variety of approaches focused on finding the best way to describe the human behaviour.

One of the most used approach was presented in [18] in order to characterize the motion of agents in the space and to find an analytical solution to prevent collisions. The main idea is related to the representation of each agent as a *Velocity Obstacle (VO)*. In particular, with the assumption related to the perfect knowledge about the shape of the obstacles and their future motion, it is possible to represent them inside the Velocity Space. The Velocity Obstacle is therefore the set of robot velocities which may cause collision with other agents. If it is possible to find a velocity outside the VO, it will be used to compute a safe trajectory.

As it is clear, a common strategy is related on taking into account the velocity of agents in order to characterize their future behaviour. In [20], an on-line approach is used to predict the space occupied by a pedestrian through the characterization of its kinematic model and its current velocity. This knowledge is used, in particular, to compute the so called *reachable set*, which includes all the future occupancies of the robot and the pedestrian in

order to avoid collisions.

In contrast with the previously defined approaches, some works in the robotic literature develop their idea under the assumption that it is not possible to have a complete knowledge about the agents inside the environment, due to the stochasticity in the pedestrian behaviour. To this aim, in [26] a method based on Deep Reinforcement Learning is presented. In particular, the work is developed under the hypothesis that it is not possible to have a priori knowledge of the human behaviour, but it is possible for a robot to learn it by simply associating weights to social norms, as for example passing to the right or keeping a safe distance, which will be updated through the training process of a neural network.

In many situations it is not possible to ignore the stochasticity of the pedestrian motion without applying a too much rough approximation. For this reason, in [1] the uncertainty of human intention is tackled from a game theoretic perspective. In particular, a set of choices is defined for each agent with a level of probability and, consequently, the computation of a collision-free path becomes the research of a Nash equilibrium.

It is possible to conclude that, by taking into account the different ways in which the human-aware navigation problem can be solved, one can choose if it is better to adopt a deterministic approach or a stochastic one.

### 2.2.2 Control Solution

The original contribution of the thesis in this particular field is related to the fact that it is possible to introduce an advanced control approach, already successfully applied in [8] for the local planning of an autonomous wheelchair inside an environment characterized by fixed obstacles or in [23] for the motion and coordination of mobile agents, called *Model Predictive Control* (MPC), in order to design the local planning of an autonomous wheelchair inside a crowded environment. The main peculiarity of this particular MPC application is related to its human-awareness.

In particular, the control approach presented in this thesis will be designed as a regulation problem in order to reach a goal position from a starting one, while ensuring the collision avoidance, with respect to fixed and moving obstacles, and also the safety and comfort requirements for the passenger of the wheelchair and, more in general, for each human inside the environment. This property can be easily achieved due to the fact that it is possible to easily include the safety requirements inside the control problem, by treating them as constraints of an optimization problem. Moreover, since the control problem will be solved inside a few seconds time interval, it is possible to highly reduce all the uncertainties related to the unpredictability of the human behaviour. For this reason, it is possible to consider a simple linear model to describe the motion of a single pedestrian and, in addition, to design a deterministic MPC, instead of a probabilistic one.

The final result of the strategy adopted in this work will be a complete motion planning approach for an indoor autonomous wheelchair inside a crowded environment, developed with respect to two different levels of control: the *RRT\* Motion Primitives* (Global) on one side and the *Model Predictive Control* (Local) on the other.

## Chapter 3

# Robot and Human Models

This chapter focuses on the characterization of the environment in which the robot will be able to perform its motion. In the first part, the model of the vehicle will be introduced through the description of the mechanical relationship between the state and control variables. Once a linearization technique is applied, the overall control scheme will be presented.

In the second part the model of a single pedestrian will be presented, in order to construct a crowded environment in which each element is fully characterized while, in the last part, in order to describe the behaviour of a crowd, a probabilistic model will be introduced.

The models of each single individual are important in the context of reactive collision avoidance and therefore are fundamental in the construction of a local planner, while the characterization of entire crowds in a given map becomes necessary in global planning.

### 3.1 Vehicle

The model presented in this work can be easily applied for a large variety of Autonomous Guided Vehicle. However, in order to justify the choices related to the tuning of parameters and, in general, to make reference to a real case, the experimental equipment used in [8] will be taken into account.

The vehicle is a motorized wheelchair, shown in Figure 3.1, produced by *Degonda Rehab SA*, endowed with two driving wheels, whose motors are characterized by a maximum power of 0.35 [kW], and three support wheels, which are used to guarantee mechanical stability.

In order to detect the relative distance with respect to near objects, the vehicle is equipped with two laser sensors *SICK TiM561*. These devices are characterized by a maximum angular amplitude of  $270^\circ$  with  $0.33^\circ$  of resolution, a maximum range of 10 [m] and a scanning frequency of 15 [Hz]. The two sensors are installed in opposite edges, in order to have a complete vision of the surrounding environment, as shown in Figure 3.2.



Figure 3.1: Wheelchair Degonda Twist t4 2x2, Image from: pg.6, Chapter 2, [8]

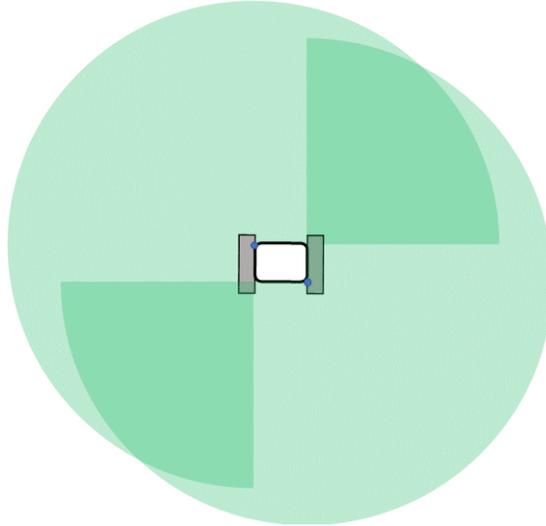


Figure 3.2: 360° Angular Range

### 3.1.1 Unicycle Model

In the context of mobile robotics it is possible to describe an AGV in different ways. The case study, related to the work done in this thesis, requires the definition of a simple model, in order to characterize the dynamic behaviour of the vehicle. To this aim, Figure 3.3 shows the representation of a unicycle model, described by the following equations:

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

where the variables  $x(t)$  and  $y(t)$  are the positions along the global refer-

ence axes of the system, while  $\theta(t)$  is the orientation of the vehicle. The longitudinal velocity  $v(t)$  and the angular speed  $\omega(t)$ , which describes the rotation around the axis perpendicular with respect to the motion plane, are the control variables of the system.

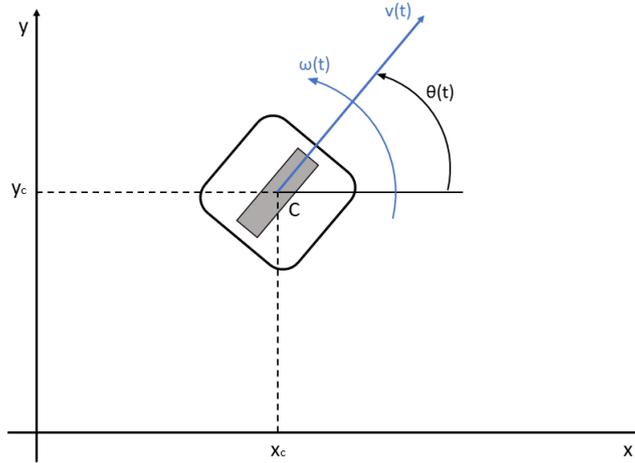


Figure 3.3: Unicycle Model

It is clear that the presented model is a cinematic one, since it relates the velocity inputs of the vehicle with the time derivatives of its configurations. Cinematic models are fundamental in order to solve the motion control problem of a mobile robot and, for this reason, in the context of planning and control, this particular choice is justified.

The intrinsic problem of the unicycle model is related to the fact that real mobile robots cannot have a single wheel, due to mechanical stability, in particular in static conditions. For this reason, a simplified model, which takes into account the definition of a two-wheeled vehicle, is introduced. This particular representation, shown in Figure 3.4, takes the name of *differential drive* and it is based on the introduction in the model of two coaxial wheels, whose actuators are independently activated.

The previous definition implies that the control variables must be expressed with respect to the angular speed of the two wheels:

$$\begin{cases} v(t) = \frac{r (\omega_R(t) + \omega_L(t))}{2} \\ \omega(t) = \frac{r (\omega_R(t) - \omega_L(t))}{d} \end{cases}$$

where  $d$  is the distance between the centre of the wheels, while  $\omega_R$  and  $\omega_L$  are the angular speeds of the right and left wheels, whose radius is represented by  $r$ .

By taking into account the wheelchair described in the previous section,

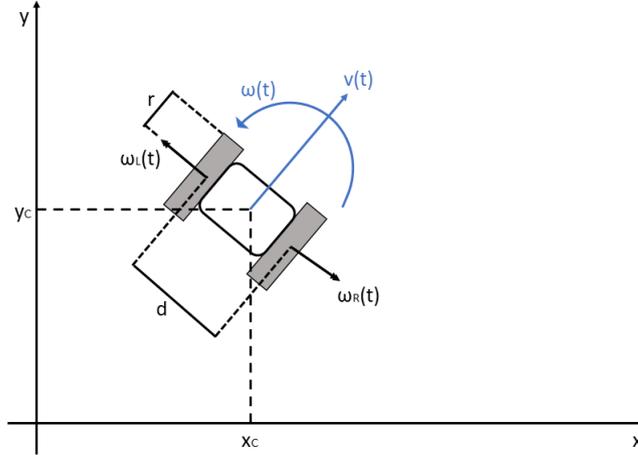


Figure 3.4: Unicycle Model with Differential Drive

it is clear that the unicycle model with differential drive easily approximates the behaviour of such vehicle. In fact, the differential drive representation allows to consider the relationship between the longitudinal and angular velocity, with respect the rotation of the two independently actuated wheels. In the same way, the real vehicle will generate a longitudinal motion by actuating the two wheels in the same direction and a rotation by actuating the two wheels in opposite directions.

### 3.1.2 Feedback Linearization

The previously described model is characterized by a non linear relationship between the state variables  $(x(t), y(t), \theta(t))$  and the control variables  $(v(t), \omega(t))$ . However, this definition is not compatible with the control requirements, due to the fact that the approach adopted in this thesis relies on a Linear MPC.

As described in [8] and [23], in the field of the control of autonomous mobile robots it is possible to approximate the vehicle as a particle, whose dynamic behaviour can be described by means of a linear relationship. This particular result can be obtained with a linearization technique, which takes the name of *Feedback Linearization*.

This approach is based on a transformation of variables, which can be obtained by considering a point P at a distance  $\varepsilon$  from the centre of the axis of the wheels, along the longitudinal direction, as highlighted in Figure 3.5.

The position P can be now expressed with respect to the global reference system:

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

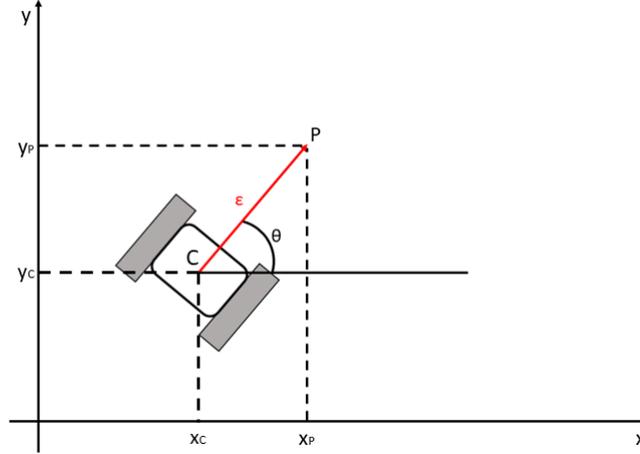


Figure 3.5: Feedback Linearization Model

where  $x_C(t)$  and  $y_C(t)$  are the global coordinates of the centre of the vehicle, around which the rotation  $\theta(t)$  is performed.

By deriving with respect to time:

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

noticing that  $\dot{x}_C(t) = v(t) \cos \theta(t)$ ,  $\dot{y}_C(t) = v(t) \sin \theta(t)$  and  $\omega(t) = \dot{\theta}(t)$ , it is possible to represent the previously defined relationship in matrix form:

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

The Feedback Linearization transformation matrix is therefore:

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

Being non-singular ( $\forall \theta$  and  $\forall \varepsilon \neq 0$ ), it is possible to invert the matrix in order to obtain the relationship between the variables of the linearized system ( $v_{Px}(t)$ ,  $v_{Py}(t)$ ) and the real one ( $v(t)$ ,  $\omega(t)$ ).

By considering now the overall control scheme, represented in Figure 3.6, it is immediately clear that the MPC will receive only the information about the position of point P, by losing every knowledge about the orientation of the vehicle, which is used instead to provide the transformation matrices.

For this reason, the MPC, which in this particular context will be applied for regulation purposes, consists only in a position control so, in the case in which a goal on orientation is required, i.e. if the desired pose differs from

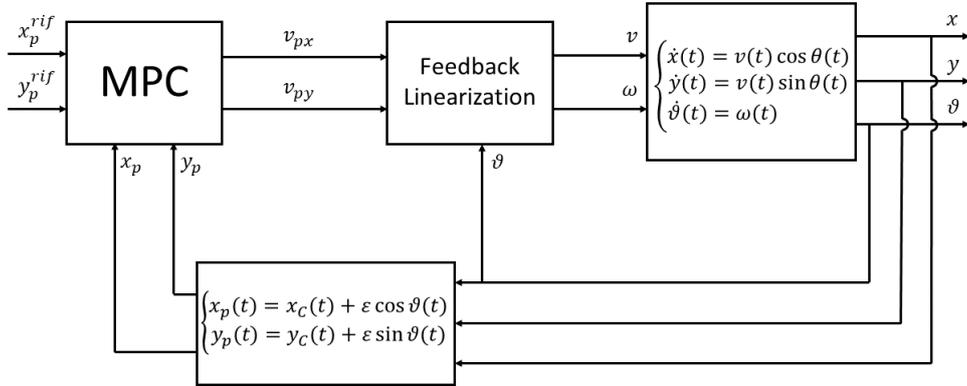


Figure 3.6: Control Scheme

the one tangent to the trajectory, a further control focused only on the orientation must be applied. The so obtained model, from the controller point of view, is therefore described by a linear decoupled system, characterized by two integrators, as shown in Figure 3.7:

$$\begin{cases} \dot{x}_P(t) = v_{Px}(t) \\ \dot{y}_P(t) = v_{Py}(t) \end{cases}$$

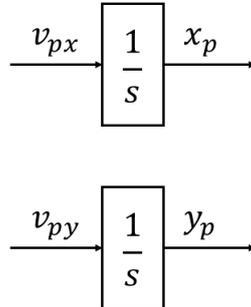


Figure 3.7: Double Integrator System

## 3.2 Pedestrian

In order to describe a human in the context of local planning, it is fundamental to fully characterize a single pedestrian as a moving obstacle.

However a second aspect, which shall not be ignored, is related to the social acceptability of the robot behaviour. For this reason, the comfort of a human, inside the environment in which the motion of the vehicle is performed, must be guaranteed. This is a crucial concept in the field of human-aware robot navigation, as it is highlighted in [24].

### 3.2.1 Kinematic of the Model

The main hypothesis, which puts the basis to the approach presented in this work, is related to the fact that the local planner will be able to predict the pedestrian motion inside a short time window. By taking into account this aspect it is possible to assume that, during this short interval, the velocity of a moving obstacle, in particular a pedestrian, will remain constant.

Therefore, in order to characterize the pedestrian kinematics, it is enough to consider a simple uniform rectilinear motion law:

$$\begin{cases} \dot{x}_{ped}(t) = v_x \\ \dot{y}_{ped}(t) = v_y \end{cases}$$

where  $\dot{x}_{ped}(t)$  and  $\dot{y}_{ped}(t)$  are the time derivatives of the pedestrian position along the axes of the global reference system, while  $v_x$  and  $v_y$  are the components of the constant pedestrian velocity.

It is important to note however that this model approximates the pedestrian as a particle. For this reason, it is assumed that the motion will be performed by the barycentre of the pedestrian body, projected on the global reference system plane, as shown in Figure 3.8.

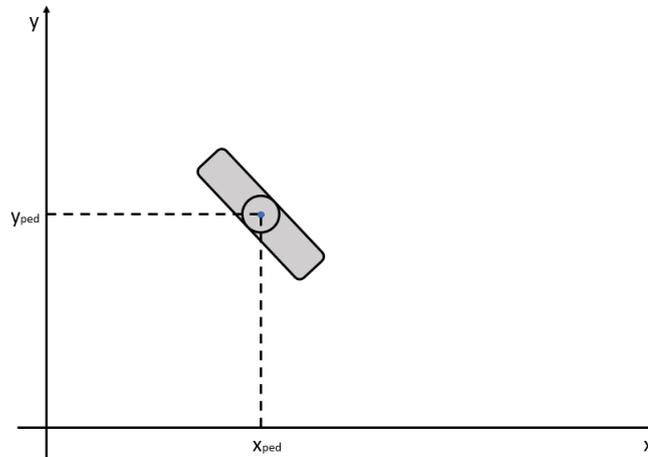


Figure 3.8: Pedestrian Model

### 3.2.2 Virtual Box

In order to take into account the size of a single pedestrian, it has been decided to include it inside a *virtual box*. This box will not only consider the measures of the human body, but also the concept of *proxemics*.

This term was coined in 1963 by the cultural anthropologist Edward T. Hall. In his work *The Hidden Dimension* ([9]), he defines the proxemics as

the study of how a human makes use of space and the effect of a crowded environment has on communication and, more in general, on social interaction. Hall described the relative distances between people, distinguishing the interpersonal distance in four zones, as shown in Figure 3.9.

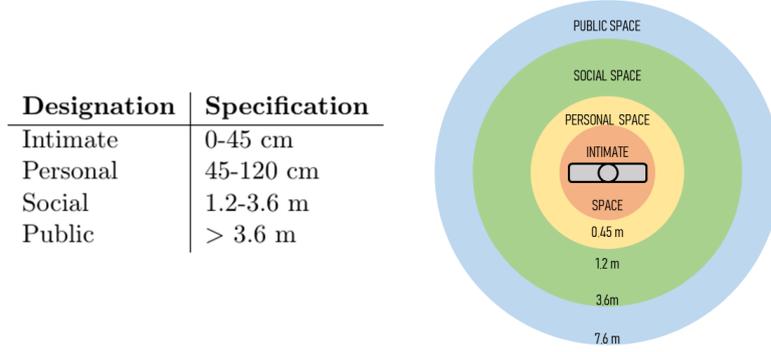


Figure 3.9: Proxemics Zones

As highlighted in [24], in order to make a person feeling safe, the robot should move outside the intimal or personal space, by avoiding therefore to approach too much closely.

Regarding the shape of the virtual box, the choice should be done with the objective to make possible for the MPC to solve a feasible problem, by avoiding non-convex sets of constraint or deadlocks in the robot trajectory.

As described in [23], the ideal obstacle avoidance can be obtained by defining a circular virtual box. However, due to the fact that such a choice is incompatible with the Linear MPC requirements, it is fundamental to make an approximation. In particular, the most convenient way is to inscribe a polytope inside the circular box. In practice, this result can be obtained by defining a set of points following a circular path. The equation governing this concept can be defined in the following way:

$$\begin{cases} x_{box}^i(t) = x_{ped}(t) + R \cos \varphi_i & \forall i = 1, \dots, n_{sides} \\ y_{box}^i(t) = y_{ped}(t) + R \sin \varphi_i & \forall i = 1, \dots, n_{sides} \end{cases} \quad (3.1)$$

where  $n_{sides}$  expresses the number of sides of a given polytope, the subscript  $i$  refers to a single point on the circular path,  $R$  is the radius of the circle defining the personal zone of a given pedestrian (whose value will be chosen by taking into account Figure 3.9) and  $\varphi_i$  is the angular sector associated to the  $i$ -th point. Figure 3.10 shows as Equation (3.1) works in practice.

The  $i$ -th side of a given polytope, inscribed inside a circle, corresponds with the  $i$ -th angular sector, which can be defined as:

$$\varphi_i = \frac{2\pi i}{n_{sides}} + \varphi_0, \quad i = 1, \dots, n_{sides}$$

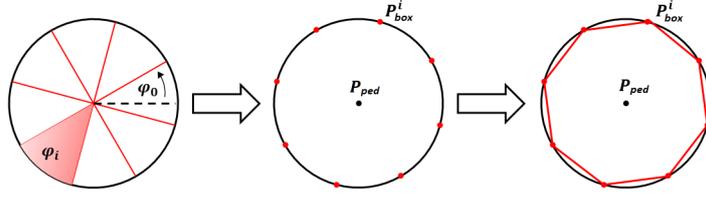


Figure 3.10: Example with  $n_{sides} = 8$

where  $\varphi_0$  expresses the initial angle, taken as starting point in order to reproduce the series of points.

It is important to note that the choice of  $\varphi_0$  will define the orientation of the polytope with respect to the global reference system. A proper choice of  $\varphi_0$  can be based on the knowledge of the pedestrian kinematics, in particular by considering its velocity vector:

$$\varphi_0 = \begin{cases} \arctan\left(\frac{v_y}{v_x}\right) & \text{if } v_x > 0 \\ \frac{\pi}{2} & \text{if } v_x = 0 \\ \arctan\left(\frac{v_y}{v_x}\right) + \pi & \text{if } v_x < 0 \end{cases}$$

where  $v_x$  and  $v_y$  are the components of the pedestrian velocity vector along the x and y directions respectively.

With this particular definition, it is possible to describe a virtual box which moves accordingly with the pedestrian trajectory.

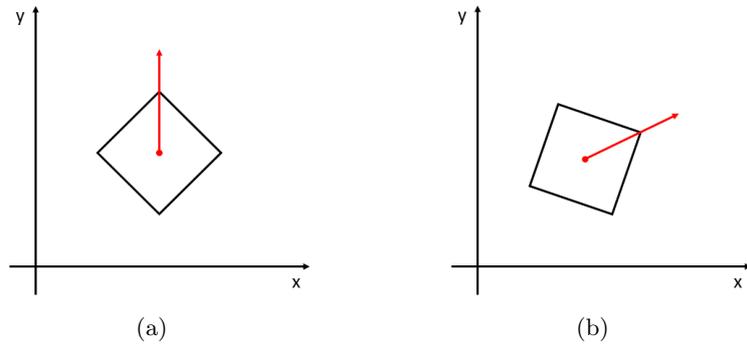


Figure 3.11: Example with  $n_{sides} = 4$ . (a)  $v_x = 0, v_y = 1$  (b)  $v_x = 2, v_y = 1$

### 3.3 Walls and Fixed Obstacles

The model described in the previous section can be also applied in the case in which the environment is characterized by the presence of fixed obstacles with a limited geometry, as for example furnitures or boxes, as shown in Figure 3.12.

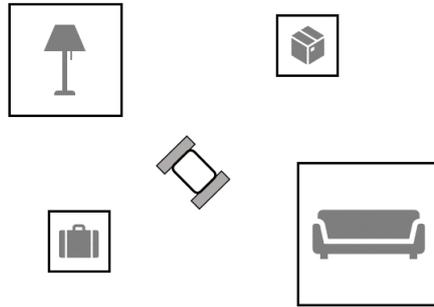


Figure 3.12: Fixed Obstacles Environment

However, the above analysis becomes not convenient for big-sized elements in the environment, as for example walls. In that case in fact, it is more convenient to keep the geometry of the obstacle unchanged, by simply reconstructing its shape through the processing of sensor data, as highlighted in Figure 3.13.

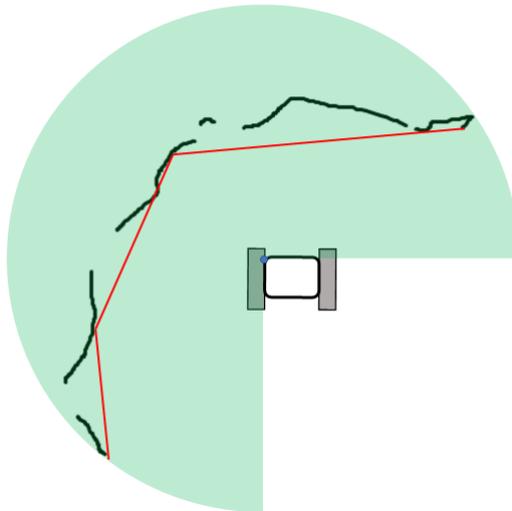


Figure 3.13: Data Reconstruction

### 3.4 Crowd

The models introduced until now, are fundamental in the field of local planning. However, in the context of path planning, a different kind of model shall be defined in order to overcome the limitations related to the global planner, as for example the lack of a reactive action. In this scenario, it is clear that a priori knowledge of the environment is needed. For this reason, the global planner will receive in input a probabilistic map entirely describing the environment by taking into account a building plan on one side, and the concentration of people on the other. This aspect plays a fundamental role in the choice of the most convenient path to compute, in order to avoid deadlocks or to overload the local planner. In fact, with this knowledge, the global planner will take into account both the presence of fixed obstacles and the characterization of human crowds and, consequently, it will compute the optimal trajectory in order to reach a predefined goal, while avoiding collisions and by discarding every path which results too much expensive in terms of distance or time.

The probabilistic maps introduced for the path planning used in this work are defined by means of clusters, each one characterized by a uniform probability distribution, as shown in Figure 3.14. Each cluster identifies a group of people, whose density will be used to compute each probability, represented with a grey level on the probabilistic map. In this section, the characterization of a crowd will be discussed.

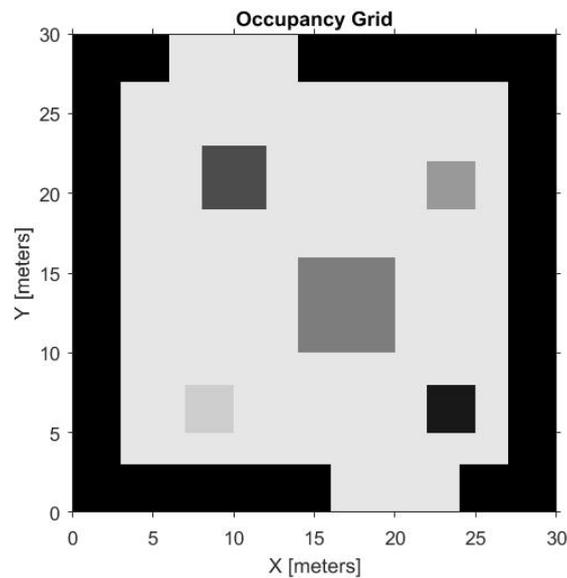


Figure 3.14: Probabilistic Map

A large number of works about the detection of people and the consequent estimation of the density of a given crowd can be found in the

literature. In general, the crowd density estimation starts from the information retrieved from external devices, in particular surveillance cameras. However, in the majority of cases, the retrieved information alone are not enough to satisfy the requirements related to safety or to risk analysis. For this reason, the acquired data need to be processed, in order to obtain a useful result.

In [15] different approaches, related to crowd density estimation from the details extracted from surveillance videos, such as background removal, information fusion or image processing and pattern recognition techniques (IP&PR), are reported. In the recent years, in order to overcome the limitations related to the crowd monitoring from videos, a more accurate approach has been proposed.

As described in [25], the key to solve the problem of crowd estimation can be found in the modern smartphones. In fact, if the devices are connected to the same Wi-fi Network, it is possible to estimate the position of a human by simply processing the informations retrieved by the MAC addresses and the exchange of packets or, more in general, passive Wi-fi signals.

This section assumes that the information about crowd density is already available and can be used to construct probabilistic maps. By taking into account the work done in [19], it is possible to estimate the crowd density by simply encapsulating each human inside a virtual cylinder, by processing the data retrieved from a surveillance camera system. The simplest way to define the density is then to consider the area covered by the range of the cameras with respect to the number of cylinders inside it, as shown in Figure 3.15.

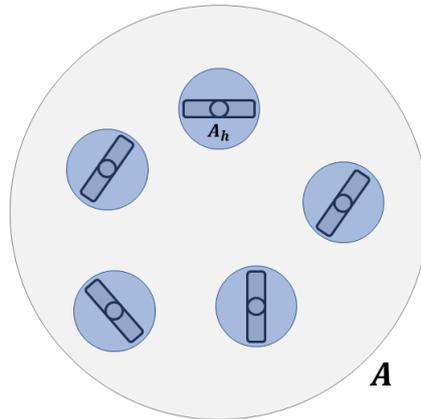


Figure 3.15: Crowd Density Model

Therefore, the instantaneous density is:

$$\rho_k = \frac{n_k}{A}$$

where  $n_k$  is the number of humans in the area  $A$  at the  $k$ -th time instant. Since a single cluster of area  $A$  is characterized by a uniform probability distribution, the probability to find occupied space inside it, at a given instant  $k$ , can be simply calculated as:

$$P_k^{occ} = \rho_k * A_h = \frac{n_k * A_h}{A}$$

where  $A_h$  is the area occupied by a single cylinder.

By observing the above definition, two main weaknesses can be noticed:

- If two cylinders are overlapped, the measure of the probability will be corrupted;
- If  $n_k * A_h > A$ , the probability is greater than one. This situation corresponds to the real case in which the area  $A$  is entirely occupied by humans and, in addition, some cylinders are not entirely contained inside  $A$ .

For this reason, in order to avoid the above defined scenarios, two hypothesis will be done:

- It is not possible for two cylinders to be overlapped, except for a negligible percentage. This result can be simply obtained by defining the cylinder diameter coincident with the length of the intimate space, previously introduced and highlighted in Figure 3.9. This statement holds under the hypothesis that each human inside the area  $A$  will not invade the intimate space of other individuals.
- The number  $n_k * A_h$  will saturate to a maximum value if the previously defined condition is true. In particular:

$$\text{if } n_k * A_h > A \text{ then } n_k * A_h = A \quad \forall k$$

With this definition, the condition  $0 \leq P_k^{occ} \leq 1$  will be always satisfied.

In conclusion, given a set of acquired data inside a discrete time interval  $[k, k + N_s]$ , the probabilistic map will be constructed by considering the average probability of each cluster, defined as:

$$\bar{P} = \frac{1}{N_s} \sum_{k=0}^{N_s} P_k^{occ}$$



## Chapter 4

# Model Predictive Control

In this chapter the approach used in the context of local planning will be described in detail.

Model Predictive Control is an advanced control method which has been introduced in the industrial field, in particular for chemical processes, since the 1980s. However, the large versatility of this class of algorithms has permitted, in the recent years, the diffusion of such approach in different fields. This particular flexibility is given by a series of characteristics, such as the possibility to explicitly include in the control problem the state and input constraints (as the saturation of the control variables or other structural limits). The formulation is based on the definition of a cost function, built by taking into account a discrete time model of the system and a finite window  $[k, k + N]$ , where  $N$  takes the name of *prediction horizon*.

### 4.1 MPC Formulation

Given a discrete time system which can be written in the form:

$$x(k + 1) = f(x(k), u(k))$$

where  $f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$ , while  $x(k) \in \mathbb{R}^n$  and  $u(k) \in \mathbb{R}^m$  are the state and the control variable respectively, by selecting a prediction horizon  $N$ , it is possible to define a performance index:

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

where  $l(x, u)$  is a suitable positive definite function which takes the name of *stage cost*, while  $V^f(x(k+N))$  is defined as *terminal cost*. In particular applications, the performance index can be written as a quadratic one:

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

where  $R = R' > 0$ ,  $Q = Q' \geq 0$  and  $S = S' \geq 0$  are weight matrices of suitable dimension associated with the control variable, the state during the prediction horizon and the final state respectively.

The idea behind the concept of predictability is related to the fact that, through the use of experimentally derived models, the knowledge of the process variables, the external measurements and the current state of the system, it is possible to predict any change in the dependent variables caused by the variations of the independent variables.

This knowledge is fundamental in order to obtain a proper set of control variables  $\mathcal{U}(k) = [u(k), \dots, u(k + N - 1)]^T$  which corresponds to the solution of the optimization problem. However, by simply following an open-loop control, i.e. applying the whole sequence of inputs, the uncertainty on the control variables will be summed up at every time instant inside the time window, increasing the overall error and causing instability.

#### 4.1.1 MPC with Receding Horizon

In order to obtain a more robust behaviour, typical of closed loop solutions, a particular technique, which goes under the name of *Receding Horizon strategy*, has been introduced.

At any time instant the optimization problem will be solved by taking into account the actual information of the process, simulating the system behaviour in the time window  $[k, k + N]$  and then, as the Receding Horizon principle suggests, only the first element  $u(k)$  of the sequence will be taken into account, as shown in Figure 4.1.

At the next time instant, the optimization problem will be solved again, this time considering the window  $[k + 1, k + N + 1]$ , by taking into account the new available system information which turn out to be updated and therefore more accurate. By analysing the recursive nature of this algorithm it becomes immediately clear that a time invariant feedback control strategy is obtained.

#### 4.1.2 Constraints

The main advantage in the use of MPC is that it is possible to directly include constraints on the state and control variables, in the form:

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

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

The previously defined constraints must be expressed in a way which is compatible with the optimization algorithm. In particular, in quadratic programming, the linear constraints can be expressed as:

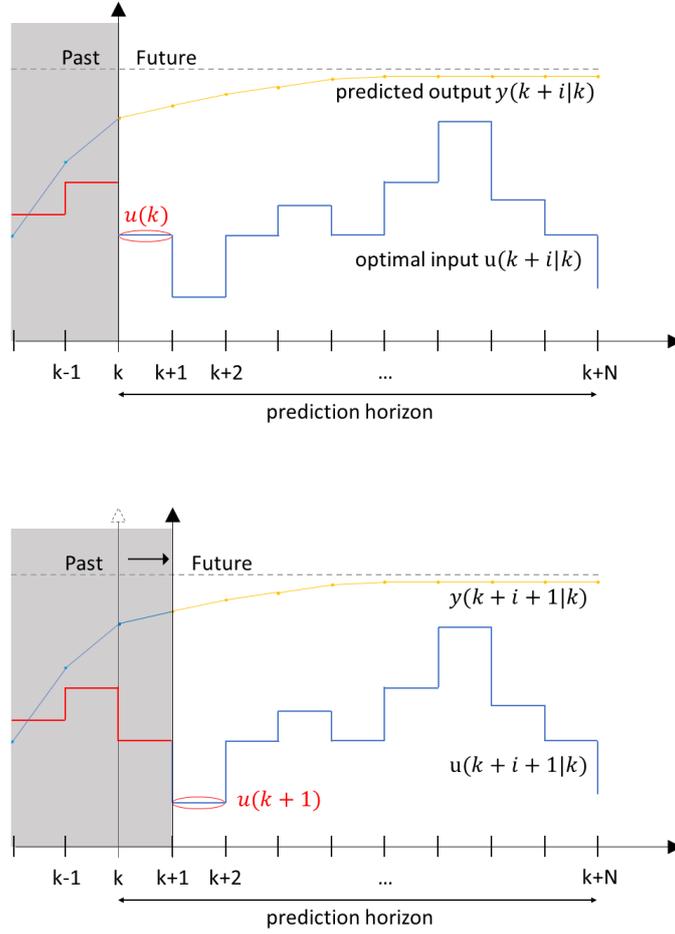


Figure 4.1: MPC with Receding Horizon

$$\begin{cases} A_{ineq_x} \mathcal{X}(k) \leq b_{ineq_x} \\ A_{eq_x} \mathcal{X}(k) = b_{eq_x} \\ \mathcal{X}_{min} \leq \mathcal{X}(k) \leq \mathcal{X}_{max} \end{cases} \quad \begin{cases} A_{ineq_u} \mathcal{U}(k) \leq b_{ineq_u} \\ A_{eq_u} \mathcal{U}(k) = b_{eq_u} \\ \mathcal{U}_{min} \leq \mathcal{U}(k) \leq \mathcal{U}_{max} \end{cases}$$

where  $A_{ineq}$  and  $A_{eq}$  are matrices whose rows express the number of conditions to be imposed,  $b_{ineq}$  and  $b_{eq}$  are vectors of constant terms, while  $\mathcal{U}_{min}$ ,  $\mathcal{X}_{min}$ ,  $\mathcal{U}_{max}$  and  $\mathcal{X}_{max}$  are the lower and the upper bounds respectively, defined for every instant of the prediction horizon. A further and more detailed analysis can be retrieved from [14].

### 4.1.3 Stability Analysis

As described in [14], in the context of the MPC, the idea of stability is strongly related to the concept of *Recursive Feasibility*, which imposes that the optimization problem, at every time instant, must ensure an acceptable solution, in order to guarantee the satisfaction of constraints and, consequently, the feasibility of the problem. To this aim, it is important to introduce the concept of *terminal set*. Given a state variable  $x(k)$  such that:

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

where  $\mathbb{X}$  is defined as *trust region* of the state, the *terminal set* can be defined as:

$$\mathbb{X}^f \subset \mathbb{X}$$

In order to ensure the solvability of the optimization problem, it is required that the final state, at the end of the prediction horizon, belongs to this particular subset of the trust region of the state:

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

The *terminal set* can be designed in order to satisfy the rules imposed by a particular control law, which takes the name of *auxiliary control law*:

$$u(k) = -K_a x(k)$$

By taking into account a generic discrete time system in the form:

$$x(k + 1) = A x(k) + B u(k)$$

the value of  $K_a$  must be chosen in a way such that the eigenvalues of the matrix  $A - BK_a$  guarantee the stability of the system. In particular, for linear systems, the value of the parameters of the control law can be found by using approaches like the LQ control or the pole placement method. Once  $K_a$  has been retrieved from one of the cited methods, by taking into account the discrete *Riccati Equation* (in which the penalty matrices Q and R are the ones used in the MPC problem):

$$(A - BK_a)^T S (A - BK_a) - S = -(Q + K_a^T R K_a) \quad (4.1)$$

it is possible to retrieve the value of the positive definite matrix S. The obtained result leads to the following statement:

$$\mathbb{X}^f = \{x(k) \mid x^T(k) S x(k) \leq \alpha\} \subset \mathbb{X}$$

where  $\alpha$  is a positive real number and

$$V^f(x) = x^T(k) S x(k)$$

is the *terminal cost*.

In order to guarantee the recursive feasibility, it is enough to require that the terminal set satisfies the property of *positive invariance*. A given set is said to be *positively invariant* if and only if:

$$x(k) \in \mathbb{X}^f \Rightarrow x(k+i) \in \mathbb{X}^f \quad \forall i = 1, \dots, N$$

with respect to the closed-loop system  $x(k+1) = (A - BK_a)x(k)$ . By recalling that  $\mathbb{X}^f \subset \mathbb{X}$ , and that  $u(k) \in \mathbb{U}$ , the following conclusion can be retrieved:

$$u(k) = -K_a x(k) \in \mathbb{U}, \quad \forall x(k) \in \mathbb{X}^f$$

This result is enough to show that if the terminal set is positively invariant, then the property of recursive feasibility is satisfied.

## 4.2 Vehicle Model for Control

As already discussed in Chapter 3, in order to obtain a model of the system suitable for a Linear MPC, the *Feedback Linearization* approach has been introduced. The resulting model consists on a linear decoupled system in the form:

$$\begin{cases} \dot{x}_P(t) = v_{Px}(t) \\ \dot{y}_P(t) = v_{Py}(t) \end{cases}$$

However, in order to make the model compatible with the discrete nature of the control approach, a further transformation is needed. By considering the discretization approach called *Forward Euler method*, such that

$$s = \frac{z-1}{\tau_{mpc}}$$

where  $\tau_{mpc}$  is the sampling time set for the controller, it is possible to obtain the discrete model of the system:

$$\begin{cases} x_P(k+1) = x_P(k) + \tau_{mpc} v_{Px}(k) \\ y_P(k+1) = y_P(k) + \tau_{mpc} v_{Py}(k) \end{cases}$$

or, in compact form:

$$\xi(k+1) = A \xi(k) + B u(k)$$

with:

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} \tau_{mpc} & 0 \\ 0 & \tau_{mpc} \end{bmatrix} \quad \xi(k) = \begin{bmatrix} x_P(k) \\ y_P(k) \end{bmatrix} \quad u(k) = \begin{bmatrix} v_{Px}(k) \\ v_{Py}(k) \end{bmatrix}$$

### 4.3 Cost Function

In order to design a cost function, suitable for the control requirements imposed by the reactive planning of the autonomous vehicle, the work done in [8] and [23] will be taken into account.

In particular, the main task of the local planner is based on reaching a given reference, which can be defined in the following way:

$$\xi_{ref} = \begin{bmatrix} x_{ref} \\ y_{ref} \end{bmatrix}$$

The goal to be reached is described by means of a desired position. This requirement can be translated, in terms of control variable, as the need for the vehicle to stop at the reference. In particular, once the vehicle reaches the desired goal, the control variable shall converge to:

$$u_{ref} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

The cost function can be then expressed in the following way:

$$J(\xi(k), u(t)) = \sum_{i=0}^{N-1} (\|\xi(k+i) - \xi_{ref}\|_Q^2 + \|u(k+i)\|_R^2) + \|\xi(k+N) - \xi_{ref}\|_S^2 \quad (4.2)$$

In order to solve the optimization problem, different strategies can be applied. A convenient way to formalize the control scheme, known as *Open-Loop Solution*, can be derived by recalling the Lagrange equation:

$$\xi(k+i) = A^i \xi(k) + \sum_{j=0}^{i-1} A^{i-j-1} B u(k+j) \quad , \quad i > 0 \quad (4.3)$$

Once a prediction horizon N is fixed, it follows that:

$$\Xi(k) = \underbrace{\begin{bmatrix} I \\ A \\ A^2 \\ \vdots \\ A^N \end{bmatrix}}_{\mathcal{A}} \xi(k) + \underbrace{\begin{bmatrix} 0 & 0 & \dots & 0 \\ B & 0 & \dots & 0 \\ AB & B & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{N-1}B & A^{N-2}B & \dots & B \end{bmatrix}}_{\mathcal{B}} u(k) \quad (4.4)$$

with

$$\Xi(k) = \begin{bmatrix} \xi(k) \\ \vdots \\ \xi(k+N) \end{bmatrix} \quad \mathcal{U}(k) = \begin{bmatrix} u(k) \\ \vdots \\ u(k+N-1) \end{bmatrix}$$

Using these variables, the performance index can be expressed in the following way:

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

where

$$\mathcal{Q} = \begin{bmatrix} Q & & & \\ & \ddots & & \\ & & Q & \\ & & & S \end{bmatrix} \quad \mathcal{R} = \begin{bmatrix} R & & \\ & \ddots & \\ & & R \end{bmatrix}$$

By recalling Equation (4.4), the cost function becomes:

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

By rearranging the previously obtained result, it is possible to express the cost function as:

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

where  $H = \mathcal{B}^T \mathcal{Q} \mathcal{B} + \mathcal{R}$  is called Hessian matrix and  $f = \mathcal{A}^T \xi^T(k) \mathcal{Q} \mathcal{B}$  is the gradient vector, while  $cost$  represents all the terms not depending from  $\mathcal{U}(k)$ . This quadratic form representation turns out to be suitable with the syntax of the most used optimization solvers.

### 4.3.1 Controller Tuning

As already introduced, matrices  $R$ ,  $Q$  and  $S$  are design parameters expressing the weights to be attributed in order to obtain a suitable control performance. In particular, the three matrices represent the penalty given to the control variables and to the error on the state and on the final state variables respectively.

As already introduced in [8] and [23], the fact that in the controller formulation a decoupled system is considered and, moreover, the fact that the model is completely symmetric, allow to design the weight matrices in the following way:

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

This result becomes clear by simply analysing matrices  $A$  and  $B$ , which turn out to be diagonal and symmetric. For this reason, it is enough to impose the same weights for both degrees of freedom. While  $q$  and  $r$  will be designed in order to guarantee a desired performance, the parameter  $s$  will be chosen in order to ensure the asymptotic stability of the system.

By taking into account the stability analysis previously discussed, the following auxiliary control law will be defined:

$$u(k) = K\xi(k)$$

where  $K$  is a symmetric diagonal matrix, such that:

$$K = \begin{bmatrix} k & 0 \\ 0 & k \end{bmatrix}$$

By solving Equation (4.1), it is possible to retrieve the relationship between the gain  $k$  and the parameter of the terminal cost:

$$s = \frac{(q + k^2 r)}{1 - (1 + \tau_{mpc} k)^2}$$

As discussed in the previous section, the gain matrix must be chosen in order to guarantee the asymptotic stability of the system. In [8], a solution, retrieved through the *Pole Placement* method, is proposed. By imposing

$$k > -\frac{2}{\tau_{mpc}}$$

the eigenvalues of the system

$$\xi(k+1) = (A + BK)\xi(k)$$

have value between 0 and 1 and so, the asymptotic stability of the system is guaranteed. However, in order to avoid oscillatory behaviours, a more conservative requirement must be imposed:

$$k > -\frac{1}{\tau_{mpc}}$$

In conclusion, it is enough to select a fraction of the previously defined limit value:

$$k = -\frac{1}{n \tau_{mpc}}, \quad \text{with } n > 1$$

## 4.4 Constraint Definition

As already introduced, the main advantage in the formulation of the MPC is based on the fact that it is possible to directly include the constraints related to the state and the control variables. This definition allows to take into account, in the research of a feasible solution, the boundaries that characterize a real context, as for example the limitations of the actuators, the presence of prohibited areas or zones occupied by obstacles and, in this particular case, comfort or safety requirements.

As described in the previous section, the formulation of a quadratic cost function allows to obtain a relationship based on the definition of the Hessian matrix and the gradient vector. In the same way, the constraints defined in this particular formulation take the form of a linear inequality:

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

where  $A_{ineq}$  is a matrix whose rows correspond to the condition imposed at a given time instant in the prediction horizon, while  $b_{ineq}$  is the vector of constant terms. It is important to note that the constraint can be also related to the state of the system and, in that case, the condition will be expressed as:

$$A_{ineq_x} \Xi(k) \leq b_{ineq_x}$$

In order to introduce in the MPC formulation also this set of constraints, it is important to express it with respect to the vector of control variables  $\mathcal{U}(k)$ .

By recalling the Lagrange equation:

$$A_{ineq_x} \underbrace{(\mathcal{A}\xi(k) + \mathcal{B}\mathcal{U}(k))}_{\Xi(k)} \leq b_{ineq_x}$$

it follows that

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

with

$$A_{ineq} = A_{ineq_x} \mathcal{B} \quad b_{ineq} = b_{ineq_x} - A_{ineq_x} \mathcal{A}\xi(k)$$

### 4.4.1 Position Constraints

By taking into account only the constraint related to the state variables:

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

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

it is possible to conclude that, since the state of the system coincides with the position of the vehicle, the trust region described through the constraint is defined as the free space of the environment in which the robot will be able to move, as shown in Figure 4.2.

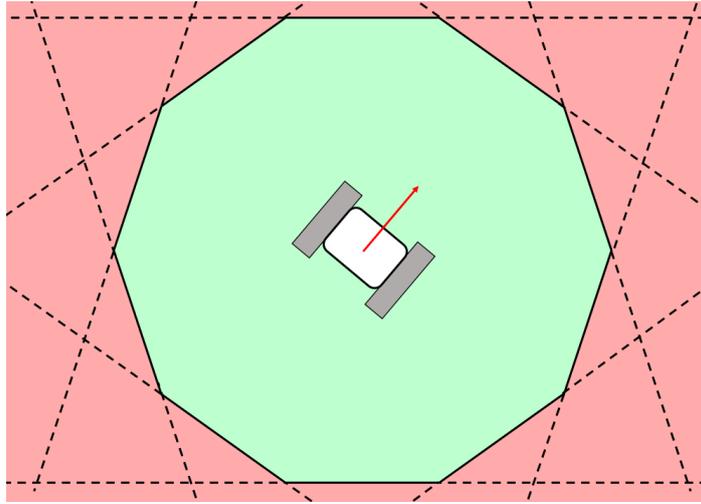


Figure 4.2: Trust Region

As it is immediately clear, by taking into account the straight lines delimiting the green area, the trust region is defined as the intersection of different half planes, whose equations take the form of:

$$h_x x + h_y y \leq l \quad (4.5)$$

where  $h_x$  and  $h_y$  are the coefficients related to the variables  $x$  and  $y$  identifying the slope of the line, while  $l$  is the constant term defining the position with respect to the origin. Each equation defined above divides the state space into two half planes, one of which describes the free space.

In addition, the presence of an obstacle inside the robot environment can be represented into the state space, by simply removing the area occupied by the obstacle from the trust region of the state, as shown in Figure 4.3.

For this reason, it is possible to use Equation (4.5) to define the geometrical shape of each given obstacle. However, as it is shown by Figure 4.3, with this representation, the trust region is not convex.

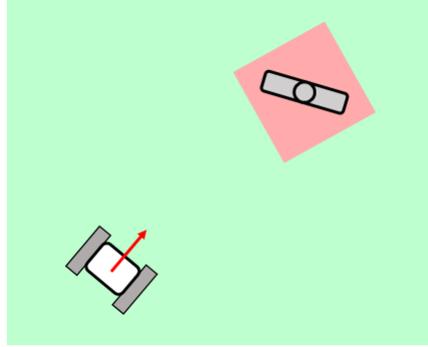


Figure 4.3: Representation of an obstacle inside the State Space

### Convexity of the Trust Region

One of the intrinsic problem in the definition of the trust region is related to the fact that it is not always possible to ensure the admissibility of the optimization problem, which is related to the convexity of the state space. This is due to:

1. The morphology of the environment. In fact it is possible for the vehicle, during the execution of its motion, to meet obstacles with a non-convex shape.
2. The errors in the data acquisition. By recalling the experimental set-up described in [8], the information about the environment is obtained through the use of sensors. For this reason it may happen that, by reconstructing the received data from external devices, non-convex geometries will arise.
3. The shape of the trust region. In fact the intersection between different half planes may be a non convex region.

While the first two statements have been solved in the previous chapter, by properly defining the shape of the virtual box, in order to solve the last problem an approach, based on the choice of the most convenient straight line to be used as a constraint, must be introduced.

### Maximum Constraint Violation Method

As already described in [8] and [23], the aim in the definition of state constraints is to ensure a trust region which is convex and large enough, compatibly with the obstacle definition.

The most used solution takes the name of *Maximum Constraint Violation Method*, represented in Figure 4.4, and it is based on the fact that, by knowing the equation describing a given obstacle, it is possible to impose

only one constraint at a time. This means that, by considering the set of straight lines surrounding a given obstacle (defined in order to construct the virtual box), it is possible to take into account only a single equation in the form of (4.5).

The choice will be done by selecting the straight line for which the corresponding constraint will be violated above all, or more simply the line such that the position of the obstacle and the actual position of the vehicle belong to different half planes.

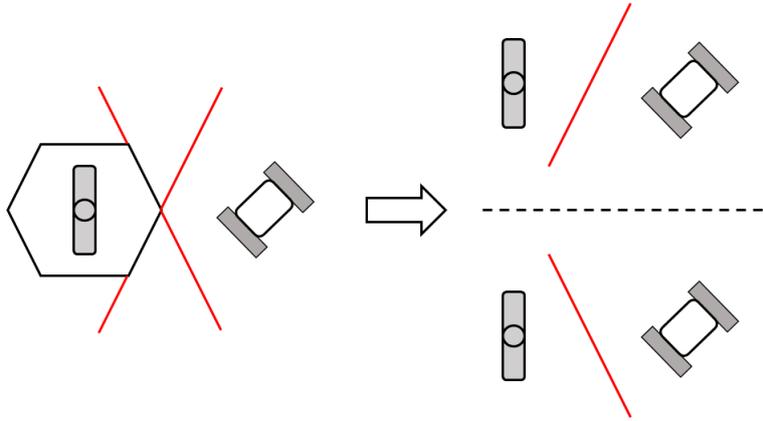


Figure 4.4: Maximum Violated Constraint

The previous definition can be formalized as follows:

$$\begin{aligned} \exists (h_x, h_y, l) : & h_x x_{pos} + h_y y_{pos} > l \wedge h_x x_{obs} + h_y y_{obs} < l \\ & \vee \\ & h_x x_{pos} + h_y y_{pos} < l \wedge h_x x_{obs} + h_y y_{obs} > l \end{aligned} \quad (4.6)$$

where  $x_{pos}$ ,  $y_{pos}$  are the components of the vehicle position with respect to the global reference frame, while  $x_{obs}$  and  $y_{obs}$  are the coordinates of the obstacle position. It is important to recall that, from the control point of view, both the vehicle and the pedestrians are seen as particles, as described in Chapter 3. For this reason,  $x_{pos}$  and  $y_{pos}$  coincide with the coordinates of the odometric centre  $C$ , introduced in Chapter 3, while  $x_{obs}$  and  $y_{obs}$  coincide with the barycentre of the pedestrian, around which the virtual box is constructed.

The only weakness of the previously defined approach is highlighted in Figure 4.5. As it is immediately clear, the two coloured lines are both candidates for the selection of the constraint, but the green one is the optimal choice because, by selecting it, the robot will find a more convenient path in terms of distance from the goal, and therefore less cost.

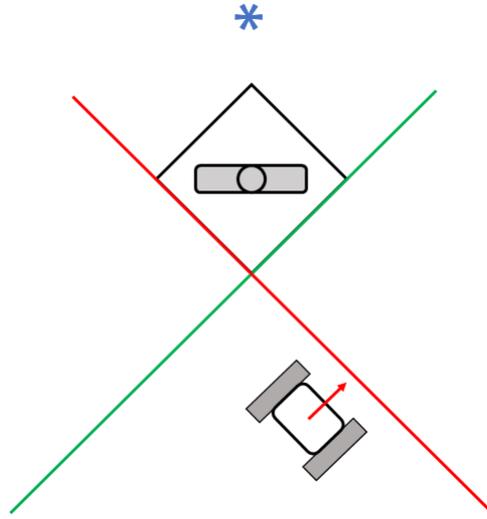


Figure 4.5: Maximum Violated Constraint: Ambiguity

In order to make possible for the robot to take the right decision, in [23] a useful approach is described. Between two lines, which are both candidates constraints for the optimization problem, the one that maximizes the distance, with respect to the vehicle position, will be taken into account, as highlighted in Figure 4.6.

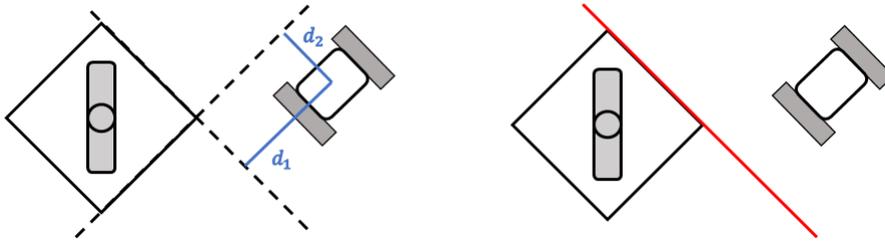


Figure 4.6: Maximum Violated Constraint: Disambiguation

### Activating Condition

In the previous section, the algorithm to select the maximum violated constraint has been described.

However it is possible to show that, even if the vehicle is very far from the obstacle, it will try to avoid it in any case. This is due to the fact that, since the virtual box surrounding a given obstacle is defined through straight lines, these will have an infinite length.

The statement defined above is clearly described in Figure 4.7. As it is shown, it is possible to see how, even if the distance between the obstacles and the vehicle is enough, the projection of the virtual box will activate the algorithm in any case.

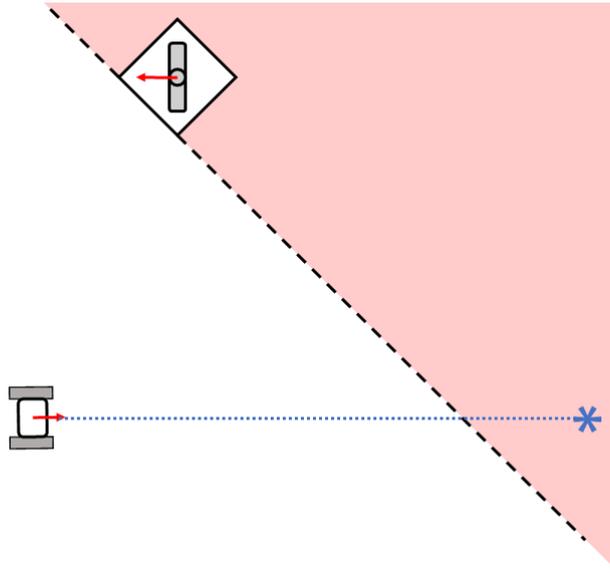


Figure 4.7: Useless Activation

In order to avoid the problem, it is possible to slightly change the activating condition, described in [8], in a way more suitable for this particular case.

The idea behind the definition is to take into account the maximum violated constraint only if the robot is dangerously approaching the pedestrian, otherwise a less strict constraint will be defined.

It is possible to formalize the above definition by means of geometric relationships. First of all, it is important to distinguish a dangerous approach from a safer one. In order to formalize the problem, a well-known solution in the field of robotics, described in [18], will be shortly introduced in the following.

Given two agents representable as circles, in order to take into account their size, and the vectors defining the direction of their velocities, it is possible to express a dangerous approaching, between the two, in the *Space of Velocity*, through the definition of *Velocity Obstacles* (VO). In order to create the model, the robot will be approximated as a particle and consequently the *Collision Cone* is defined. Such a cone can be built by simply considering the radius of the circle surrounding the obstacle, as shown in Figure 4.8.

The collision cone  $CC_{R,obs}$  is then evaluated with respect to the relative velocity between the robot and the moving obstacle. In particular, if the

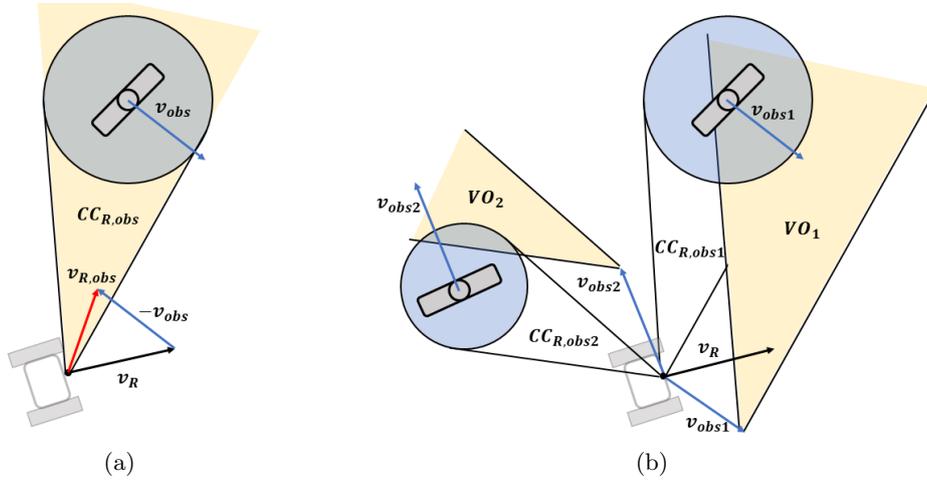


Figure 4.8: (a) Collision Cone and Velocity Obstacle in a Single-Agent Framework (b) Velocity Obstacles in a Multi-Agent Framework

intersection between the set of relative trajectories and the area defined by the radius of the circle is different from zero, then a collision will happen. In the case in which the robot will perform its motion in a multi-agent system, it is more convenient to define the VO by translating each cone along the direction of the velocity vector of each obstacle. However, as it will be described in the next sections, the MPC formulation will take into account each obstacle independently, by considering then the union of the constraints related to them and, for this reason, a single agent problem can be solved for each constraint. In order to define a dangerous approach, it is necessary to consider the segment of the virtual box lying on the line coincident with the Maximum Violated Constraint as shown in Figure 4.9.

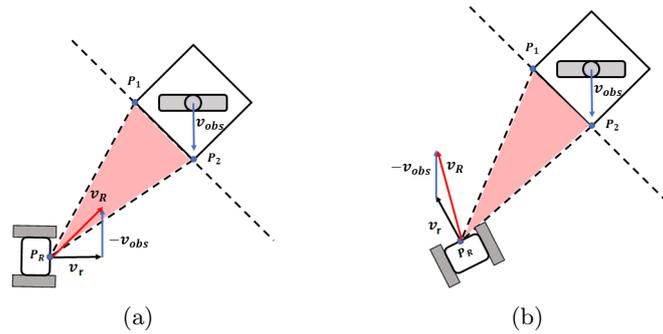


Figure 4.9: Vehicle Motion: (a) Dangerous Approach (b) Safe Approach

The collision cone is defined by the two lines connecting the position of the robot and the two extremities of the segment. An approach is then

defined dangerous if the vector describing the relative velocity is found inside the area defined by the collision cone (red area). In order to formalize the previous statement, it is possible to recall an important property of vectors from linear algebra. Figure 4.10 represents three vectors, defined as:

$$\vec{v}_1 = \begin{bmatrix} P_1.x - P_r.x \\ P_1.y - P_r.y \end{bmatrix} \quad \vec{v}_2 = \begin{bmatrix} P_2.x - P_r.x \\ P_2.y - P_r.y \end{bmatrix} \quad \vec{v}_R = \vec{v}_r - \vec{v}_{obs} = \begin{bmatrix} v_r.x - v_{obs}.x \\ v_r.y - v_{obs}.y \end{bmatrix}$$

where  $\vec{v}_r$  and  $\vec{v}_{obs}$  are the vectors defining the direction of the velocities of the robot and the obstacle respectively, while  $P_1$  and  $P_2$  are the points defining the extremities of the segment of the virtual box.

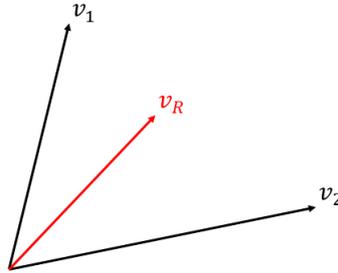


Figure 4.10: Vector Definition

The vector  $\vec{v}_R$  can be written as a linear combination of the two vectors defining the red area:

$$\vec{v}_R = \alpha_1 \vec{v}_1 + \alpha_2 \vec{v}_2 \quad (4.7)$$

where  $\alpha_1$  and  $\alpha_2$  are two constant. With this notation, it is possible to analyse the direction of the vector  $\vec{v}_R$  with respect to the red area by simply looking at the values of  $\alpha_1$  and  $\alpha_2$ , in particular by looking at their signs.

In fact, as highlighted by Figure 4.11, four configurations may be obtained. In particular, the case in which the two scalars are positive coincides with the situation in which the vector  $\vec{v}_R$  is inside the red area.

By rewriting (4.7) with respect to the global reference axes, the following system of equations is retrieved:

$$\begin{cases} v_{Rx} = \alpha_1 v_{1x} + \alpha_2 v_{2x} \\ v_{Ry} = \alpha_1 v_{1y} + \alpha_2 v_{2y} \end{cases}$$

It is important to note that, since the three vectors are known quantities, the system should be written highlighting the vector  $\vec{\alpha}$ . In matrix form, it becomes:

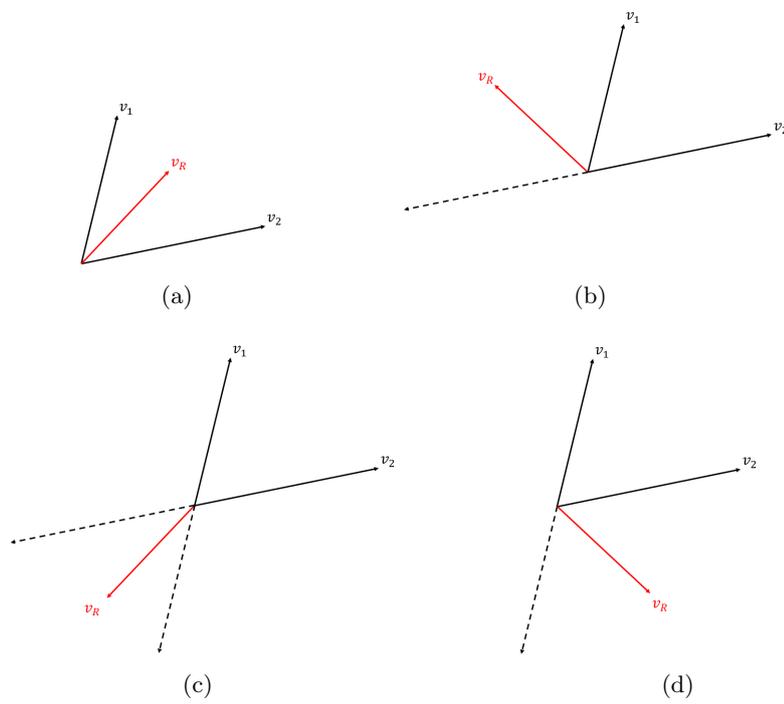


Figure 4.11: Direction of  $v_R$ : (a)  $\alpha_1 > 0$ ,  $\alpha_2 > 0$  (b)  $\alpha_1 > 0$ ,  $\alpha_2 < 0$  (c)  $\alpha_1 < 0$ ,  $\alpha_2 < 0$  (d)  $\alpha_1 < 0$ ,  $\alpha_2 > 0$ .

$$\begin{bmatrix} v_{Rx} \\ v_{Ry} \end{bmatrix} = \begin{bmatrix} v_{1x} & v_{2x} \\ v_{1y} & v_{2y} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix}$$

Therefore:

$$\vec{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = [V]^{-1} \vec{v}_R$$

$$\text{with } [V]^{-1} = \frac{1}{v_{1x} v_{2y} - v_{2x} v_{1y}} \begin{bmatrix} v_{2y} & -v_{2x} \\ -v_{1y} & v_{1x} \end{bmatrix}.$$

However, the matrix  $[V]$  may be singular, in particular for  $v_{1x} v_{2y} = v_{2x} v_{1y}$ . This result may be interpreted geometrically, by distinguishing different scenarios:

- $\vec{v}_1 = \vec{v}_2$ . This case coincides with the situation in which the segment  $\overline{P_1 P_2}$  degenerates into one point.
- $v_{1x} = v_{1y} \wedge v_{2x} = v_{2y}$ . The two vectors are coincident or the angle between them is equal to 0 or  $\pi$ .
- $v_{1x} = v_{1y} = 0 \vee v_{2x} = v_{2y} = 0$ . This case is related to the situation in which the position of the vehicle coincides with one of the extremes of the segment.
- $v_{1x} = v_{2x} = 0 \vee v_{1y} = v_{2y} = 0$ . The two vectors are parallel, which is impossible since they have a point in common (the position of the vehicle).

By observing the above described scenarios it is possible to see that, in the majority of cases, the singularity of matrix  $[V]$  is simply avoided by construction. In fact, the geometrical definition of the virtual box allows to avoid every scenario in which the segment degenerates into a point, or when the angle formed by the vectors is equal to 0 or  $\pi$ . In the case in which, from the definition of the box, the matrix turns out to be non-invertible, it is enough to automatically define the approach of the vehicle as a dangerous one.

Once  $\vec{\alpha}$  has been retrieved, by simply checking the sign of its components, it is possible to discover if  $\vec{v}_R$  is contained in the red area and, therefore, if the robot is dangerously approaching the pedestrian. As previously introduced, if the statement defined above is satisfied, then the line obtained by applying the maximum violated constraint approach will be taken into account.

Conversely, if the robot direction appears to be safe for a pedestrian, i.e. if  $\vec{v}_R$  is found outside the red area, then a different constraint will be chosen.

In particular, the new constraint will be computed by taking into account the line perpendicular with respect to the one joining  $P_r$  and the nearest point between  $P_1$  and  $P_2$ :

$$\begin{cases} h_x = P_{min}.x - P_r.x \\ h_y = P_{min}.y - P_r.y \\ l = h_x P_{min}.x + h_y P_{min}.y \end{cases}$$

with  $P_{min} = \arg \min_{i=1,2} d(P_r, P_i)$ .

Figure 4.12 shows the two different scenarios.

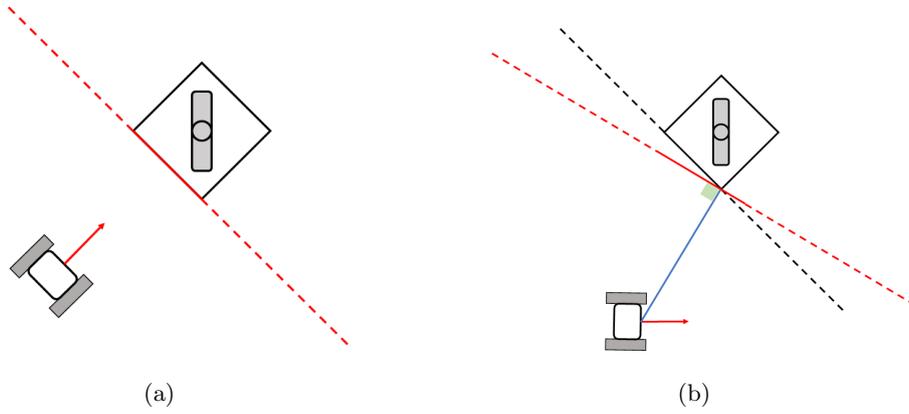


Figure 4.12: Activating Condition: (a) Dangerous Approach (b) Safe Approach

### Number of Sides of the Virtual Box

As previously described, the convexity of the trust region is highly dependent from the choice of the number of sides of the virtual box.

Many works in literature tend to use a high number of sides, in order to reduce the probability to find an unfeasible solution. In fact, as highlighted in [23], the best choice in order to represent obstacles inside the the operative space of the robot is related to the use of a circular shape, which however does not allow to obtain a convex region. For this reason, the definition of a convex polytope with a high number of sides is the one that better approximate a circle, while guaranteeing a feasible solution. In this particular approach, the number of sides of the polytope has been reduced to four. This choice, although it may seem too much rough, is justified by the fact that the orientation of the virtual box changes with respect to the direction of the pedestrian velocity vector, in particular one edge of the geometric figure points in the same direction of the obstacle velocity vector.

Even if the definition above covers the majority of cases, it may happen that, for particular configurations, the vehicle will fall into a deadlock although a free path is available. In particular, Figure 4.13 shows the worst cases.

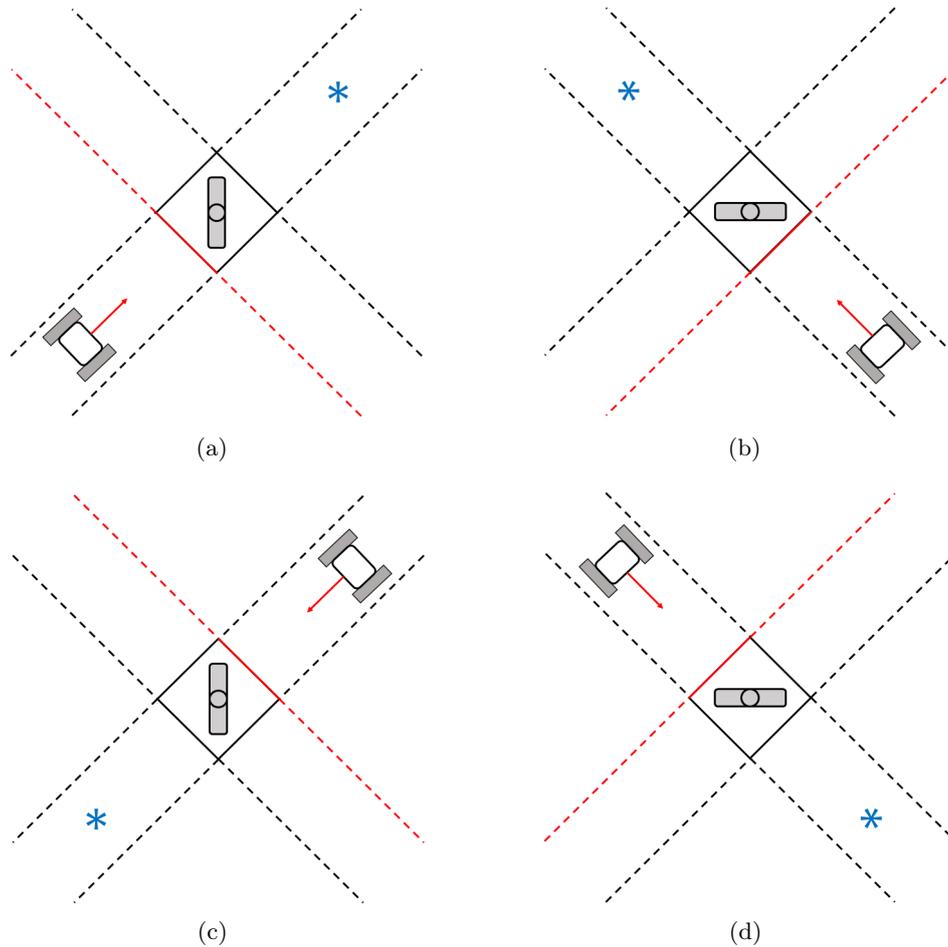


Figure 4.13: Worst Cases: the vehicle doesn't reach the goal, instead it stops in front of the obstacle

It is important to note however that, by considering the approximation under which one of the edges of the virtual box denotes the direction of the velocity vector, even if for some time instants the goal may be hidden by the obstacle, inside the prediction horizon the obstacle will show a free path with its motion, as shown in Figure 4.14.

The configuration in Figure 4.13 becomes problematic when the pedestrian velocity is close to zero. In this case in fact it may happen that the obstacle will hide the free path during the entire prediction horizon. In order to solve the situation, a simple approach will be introduced.

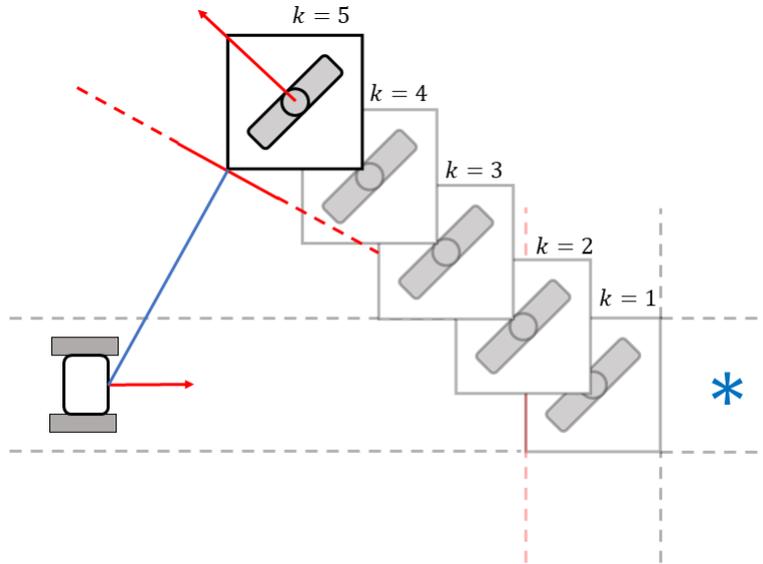


Figure 4.14: Moving Obstacle

For steady state obstacles with a priori fixed dimension and for small speed moving pedestrians it is not always possible to define the orientation of the virtual box, since the velocity vector may be equal to zero in modulus. The orientation will be then chosen by taking into account the vector defining the direction of the line joining the position of the robot and the centre of the virtual box, as highlighted in Figure 4.15.

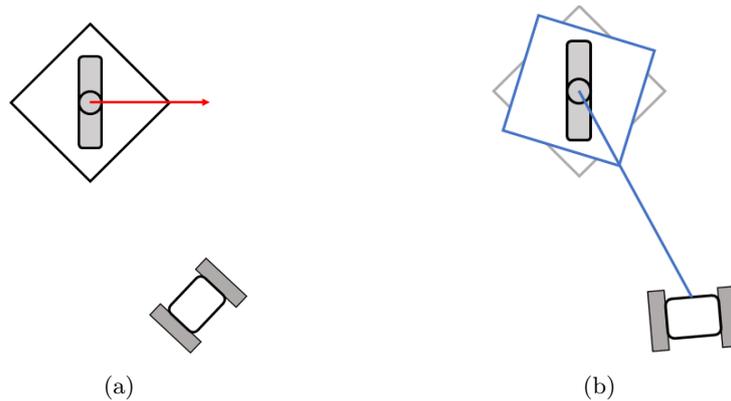


Figure 4.15: Initial Orientation of the Virtual Box: (a)  $|v_r| > 0$  (b)  $|v_r| \simeq 0$

As it is immediately clear, by applying the simple rule defined above, the worst case, caused by the uncertainty in the orientation of the virtual box for velocities around zero, will be avoided.

### Pedestrian Model for Control

As previously described, the need to confer robustness to the overall execution of the MPC, implies the introduction of the Receding Horizon technique. In this particular context, an advantageous choice consists in the definition of a short time window, in which the control strategy will be applied.

The use of a dynamic window approach, like the Receding Horizon, highly reduces the uncertainty related to the pedestrian motion. In fact, by selecting a small value of  $N$ , it is possible to obtain a few seconds time window.

With this definition, the pedestrian motion, in a short time interval, can be considered linear, while its velocity space is characterized by a constant speed. By recalling the kinematic model introduced in the previous chapter:

$$\begin{cases} \dot{x}_{ped}(t) = v_x \\ \dot{y}_{ped}(t) = v_y \end{cases}$$

where  $\dot{x}_{ped}(t)$  and  $\dot{y}_{ped}(t)$  are the time derivatives of the pedestrian position along the axes of the global reference system, while  $v_x$  and  $v_y$  are the components of the constant pedestrian velocity, it is possible to introduce again a discretization approach in order to obtain a model compatible with the MPC formulation. The discrete time representation is then formalized as:

$$\begin{cases} x_{ped}(k+1) = x_{ped}(k) + \tau_{mpc} v_x \\ y_{ped}(k+1) = y_{ped}(k) + \tau_{mpc} v_y \end{cases}$$

where  $\tau_{mpc}$  is the sampling time of the controller.

The introduced representation allows to consider the pedestrian motion by means of a segmented trajectory, as shown in Figure 4.16. In this scenario, at a given frequency, the controller will receive a point of the discretized trajectory which will identify the pedestrian position at each time instant.

In order to make possible for the MPC to simulate the pedestrian behaviour, it is assumed that the controller receives the information about the velocity of each obstacle in the environment.

This assumption, although it may seem weak, can be however validated by the fact that the actual velocity of a body, whose description comes from the sensor data, can be estimated by taking into account the information about the pedestrian position available during the previous time step. In particular:

$$v(k) = \frac{x_{ped}(k) - x_{ped}(k-1)}{\tau_s}$$

where  $\tau_s$  is the sampling time of the position measurements.

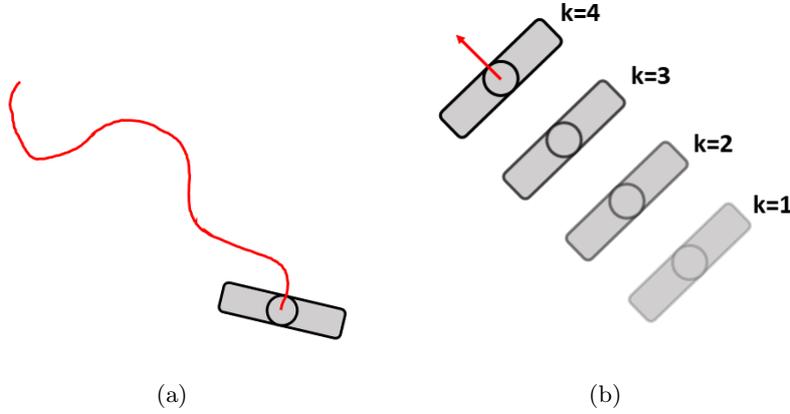


Figure 4.16: Pedestrian Trajectory: (a) Unpredictable Motion (b) Linear Motion in a small discrete time window

### State Constraint Implementation

As already discussed, the constraint related to the position can be defined as a linear combination between the coordinates expressed with respect to the global reference frame formalized as:

$$h_x^{(i)}x(k+i) + h_y^{(i)}y(k+i) \leq l^{(i)} \quad i = 0, \dots, N \quad (4.8)$$

The above equation defines a constraint for each position of the vehicle inside the prediction horizon. This conclusion is quite trivial since, as shown in Figure 4.17, by performing its motion, the vehicle may overcome a given obstacle. At the same time instant, the obstacle will show a different side of the virtual box. This means that, inside a prediction horizon, different constraints can be defined.

In addition, in order to express the state constraint through a representation compatible with the MPC formulation, the following matrix relationship must be introduced:

$$\begin{bmatrix} h_k & & \\ & \ddots & \\ & & h_{k+N} \end{bmatrix}_{(N+1, 2(N+1))} \begin{bmatrix} \xi(k) \\ \vdots \\ \xi(k+N) \end{bmatrix}_{(2(N+1), 1)} \leq \begin{bmatrix} l_k \\ \vdots \\ l_{k+N} \end{bmatrix}_{(N+1, 1)} \quad (4.9)$$

Equation (4.9) extends the position constraints to the overall prediction horizon  $[k, k+N]$ . The above statement can be read as the need for the control strategy to guarantee the non-violation of the constraints during the overall execution.

It is important to note that (4.9) is expressed with respect to the compact

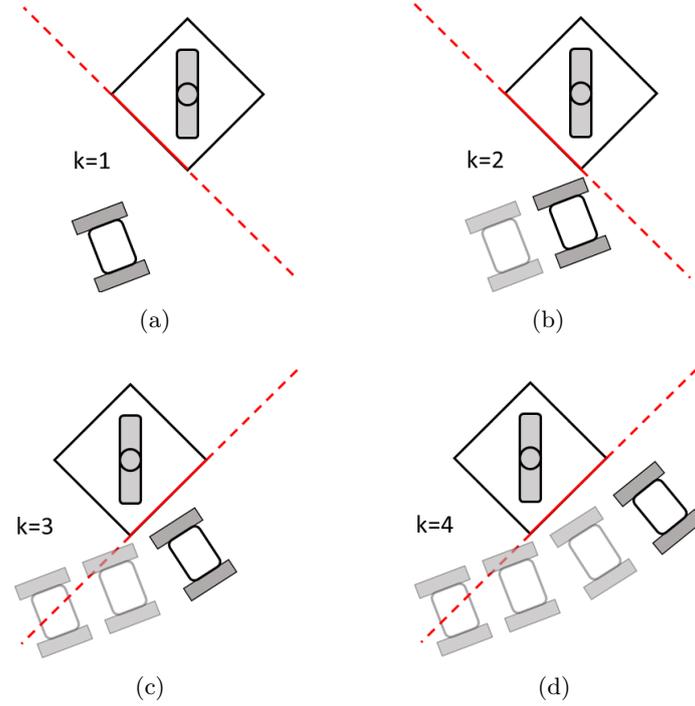


Figure 4.17: Motion inside a prediction horizon

form of Equation (4.8), i.e.

$$\underbrace{\begin{bmatrix} h_x^{(i)} & h_y^{(i)} \end{bmatrix}}_{h_{k+i}} \xi(k+i) \leq l_{k+i} \quad i = 0, \dots, N$$

Another important aspect of the matrix expression defined previously is that the subscripts  $k, \dots, k+N$  reinforce the fact that the constraints are computed not only with respect to the actual position of the vehicle, but also for the future ones.

However, at every time instant, the robot can access only its actual position, expressed as:

$$\xi(k) = \begin{bmatrix} x_{pos}(k) \\ y_{pos}(k) \end{bmatrix}$$

In order to overcome this problem, it is possible to recall the *Lagrange Equation*:

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

The vector

$$\Xi(k) = \begin{bmatrix} \xi(k) \\ \vdots \\ \xi(k+N) \end{bmatrix}_{(2(N+1),1)}$$

represents all the future positions of the vehicle which can be reached if each element of the entire vector of the control variables  $\mathcal{U}(k)$  will be applied at the corresponding time instant. However, the introduction of the Receding Horizon strategy imposes, in order to guarantee robustness, to discard all the elements of  $\mathcal{U}(k)$  except for the first one, which will be imposed to the system.

It becomes now clear that, by assuming the availability of  $\mathcal{U}(k-1)$  it is possible to obtain all the future positions of the vehicle, with respect to the previous time instant:

$$\Xi(k-1) = \mathcal{A}\xi(k-1) + \mathcal{B}\mathcal{U}(k-1)$$

where it is assumed that the robot will store the value of the previous position  $\xi(k-1)$ . For sake of clarity, Figure 4.18 introduces a control scheme and the corresponding response of the system inside the state space, simulated by the MPC in order to compute the constraints.

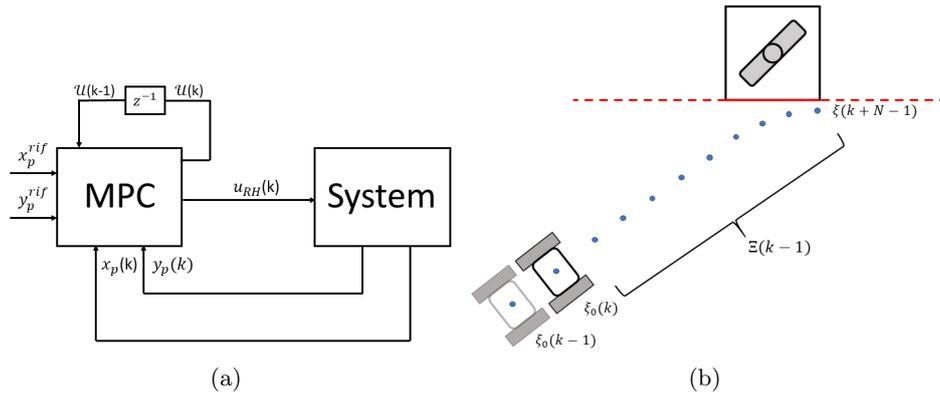


Figure 4.18: 4.18(a) The vector  $\mathcal{U}(k)$  is not use directly on the control, but only to compute the future positions 4.18(b) This is not the true response of the system, but only the theoretical one, computed by the MPC in order to obtain the space constraints.

It is now possible to rewrite (4.9) by taking into account this aspect:

$$\underbrace{\begin{bmatrix} h_{k|k-1}^{(x_0, y_0)} \\ \vdots \\ h_{k+N|k-1}^{(x_N, y_N)} \end{bmatrix}}_{H_{obs}} \underbrace{\begin{bmatrix} \xi(k) \\ \vdots \\ \xi(k+N) \end{bmatrix}}_{\Xi(k)} \leq \underbrace{\begin{bmatrix} l_{k|k-1}^{(x_0, y_0)} \\ \vdots \\ l_{k+N|k-1}^{(x_N, y_N)} \end{bmatrix}}_{L_{obs}} \quad (4.10)$$

In this case, the subscripts  $k|k-1, \dots, k+N|k-1$  express the fact that the constraint will be computed for every time instant in the prediction horizon, by taking into account the information retrieved in the previous execution. Moreover, the superscripts  $(x_i, y_i)$  represent the situation in which, since the problem mainly deals with pedestrians, the constraint may change not only due to the motion of the robot, but also with respect to the motion of the obstacle, as highlighted in Figure 4.19.

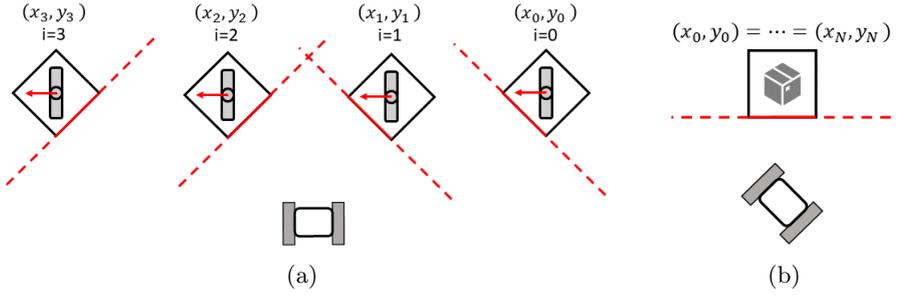


Figure 4.19: 4.19(a) Moving Obstacle (Pedestrian) 4.19(b) Fixed Obstacle (Box)

In particular, by recalling the pedestrian kinematic model, the positions  $(x_i, y_i)$  can be retrieved as:

$$\begin{cases} x_{i+1} = x_i + \tau_{mpc} v_x \\ y_{i+1} = y_i + \tau_{mpc} v_y \end{cases} \quad \forall i = 0, \dots, N-1$$

with  $(x_0, y_0) = (x_{obs}, y_{obs})$ .

in the case in which the constraint must be defined with respect to a fixed obstacle, Equation (4.10) can be easily simplified by imposing:

$$\begin{aligned} x_{i+1} &= x_i \\ y_{i+1} &= y_i \end{aligned} \quad \forall i = 0, \dots, N-1$$

In conclusion, (4.10) can be expressed in compact form as:

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

However, it is important to check if the half plane denoting the free space is the one in which the estimated position of the vehicle is contained. This condition can be verified by imposing, for each constraint, the following relationship:

$$h_x^{(i)} \hat{x}_c + h_y^{(i)} \hat{y}_c \leq l^{(i)} \quad i = 0, \dots, N$$

where  $\hat{x}_c$  and  $\hat{y}_c$  are the coordinates of the vehicle estimated position. In the case in which the condition is not satisfied, i.e.

$$h_x^{(i)} \hat{x}_c + h_y^{(i)} \hat{y}_c \geq l^{(i)} \quad i = 0, \dots, N$$

in order to obtain an inequality with the lower or equal sign, it is necessary to invert the sign of each parameter:

$$\left(-h_x^{(i)}\right) \hat{x}_c + \left(-h_y^{(i)}\right) \hat{y}_c \leq \left(-l^{(i)}\right) \quad i = 0, \dots, N$$

The last step, before sending the computed matrices to the controller, is to make this constraint on the state variables suitable for the MPC formulation. Therefore, it is fundamental to express it with respect to the vector of control variables. By recalling again the Lagrange equation:

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

from which

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

### Complete Constraints

The previously defined equation can be written in compact form as:

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

with

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

However, as the subscript suggest, the constraint refers only to a single obstacle. In fact, in order to guarantee a precise collision avoidance in a crowded environment, the previously introduced analysis must be done for every single obstacle, in order to consider the union of the different trust regions, as shown in Figure 4.20.

By defining  $A_{obs}^{(j)}$  and  $b_{obs}^{(j)}$  as the constraint related to the  $j$ -th obstacle, it is possible to build the resulting trust region by simply defining:

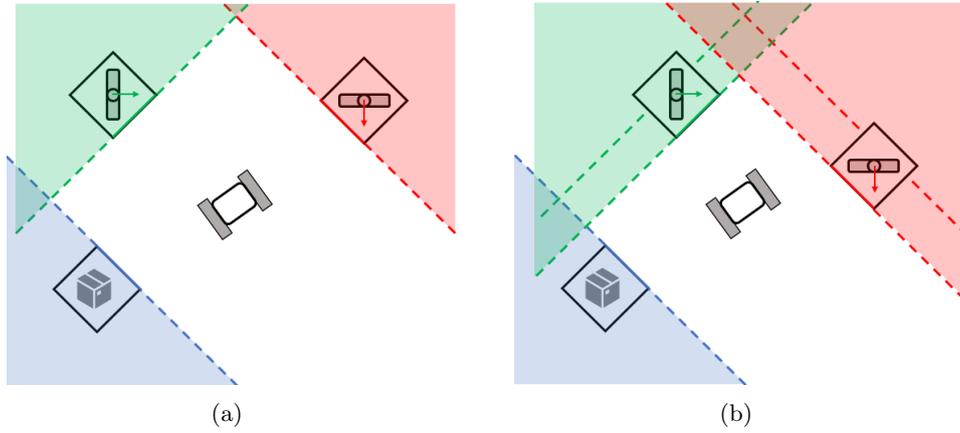


Figure 4.20: The trust region is obtained by removing the coloured areas 4.20(a) Time Instant  $i$  4.20(b) Time Instant  $i+1$

$$\begin{bmatrix} A_{obs}^{(1)} \\ \vdots \\ A_{obs}^{(n_{obs})} \end{bmatrix}_{(n_{obs}*(N+1), 2N)} \quad \begin{bmatrix} b_{obs}^{(1)} \\ \vdots \\ b_{obs}^{(n_{obs})} \end{bmatrix}_{(n_{obs}*(N+1), 1)}$$

#### 4.4.2 Velocity Constraint

One of the main problems associated with control approaches is related to the fact that, even if a solution appears to be feasible with respect to the requirements imposed by the problem, it may happen that, in order to perform the desired behaviour, the controller will apply values of control variables incompatible with the intrinsic limitations of the system. In this particular analysis, it may happen that, even if a feasible solution for the MPC problem exists, the velocity requested to the motor of the vehicle will be higher with respect to its limitations. It is clear that the above requirements can be expressed as a saturation in velocity or, more in detail, by means of constraints on the control variables.

In order to justify the choice in the parameters, the model of the wheelchair will be taken into account and, consequently, the work described in [8] will be highlighted. As already discussed in the cited document, since the considered system is a decoupled one, it is not possible to impose constraints directly on the velocities along the two directions of the axes of the global reference frame. This is due to the fact that, by imposing limits along the two directions, the longitudinal velocity will not be limited in a uniform way, in fact the maximum value of the velocity will change with respect to the

motion direction. This behaviour becomes incompatible with the requirements described above.

For this reason, it is fundamental to impose the constraints on the longitudinal direction, in the following way:

$$-v_{max} \leq v_{long} \leq v_{max}$$

where  $v_{max}$  will be selected in order to satisfy the limitations imposed by the system and, compatibly, the requirements related to the comfort.

The relationship between the modulus of the longitudinal velocity and the decoupled ones is however non linear:

$$|v_{long}| = \sqrt{v_x^2 + v_y^2} \leq v_{max}$$

In order to obtain a linear relationship, the first order expansion of the Taylor series is considered:

$$f(v_x, v_y) = f(\bar{v}_x, \bar{v}_y) + \frac{\partial f(\bar{v}_x, \bar{v}_y)}{\partial v_x} (v_x - \bar{v}_x) + \frac{\partial f(\bar{v}_x, \bar{v}_y)}{\partial v_y} (v_y - \bar{v}_y)$$

where  $f(v_x, v_y) = \sqrt{v_x^2 + v_y^2}$  and the point  $(\bar{v}_x, \bar{v}_y)$  is obtained by considering the velocities from the previous time instant. It follows that:

$$0 \leq \sqrt{\bar{v}_x^2 + \bar{v}_y^2} + \frac{1}{2\sqrt{\bar{v}_x^2 + \bar{v}_y^2}} (2\bar{v}_x(v_x - \bar{v}_x) + 2\bar{v}_y(v_y - \bar{v}_y)) \leq v_{max}$$

which becomes

$$0 \leq \frac{\bar{v}_x}{\sqrt{\bar{v}_x^2 + \bar{v}_y^2}} v_x(k+i) + \frac{\bar{v}_y}{\sqrt{\bar{v}_x^2 + \bar{v}_y^2}} v_y(k+i) \leq v_{max}$$

$$\forall i \in \{0, \dots, N-2\}$$

The main problem in the above statement is that, since the value of the denominator cannot be equal to zero, this means that it is not possible to apply the constraint for small velocities. The adopted solution in this particular situation is based on imposing a different set of constraints.

To this aim, the selected constraint acts on the two decoupled velocities  $v_x$  and  $v_y$ :

$$-v_{max} \leq v_x(k+i) \leq v_{max}$$

$$-v_{max} \leq v_y(k+i) \leq v_{max}$$

$$\forall i \in \{0, \dots, N-2\}$$

In order to make the definition compatible with the MPC formulation, it is necessary to extend the constraints along the overall prediction horizon. In particular, for high velocities:

$$A_v = \begin{bmatrix} \frac{\bar{v}_x}{\bar{v}} & \frac{\bar{v}_y}{\bar{v}} & & & & \\ & & \ddots & & & \\ & & & \frac{\bar{v}_x}{\bar{v}} & \frac{\bar{v}_y}{\bar{v}} & \\ -\frac{\bar{v}_x}{\bar{v}} & -\frac{\bar{v}_y}{\bar{v}} & & & & \\ & & \ddots & & & \\ & & & -\frac{\bar{v}_x}{\bar{v}} & -\frac{\bar{v}_y}{\bar{v}} & \end{bmatrix}_{(2N,2N)} \quad b_v = \begin{bmatrix} v_{max} \\ \vdots \\ v_{max} \\ \Delta v_{max} \\ 0 \\ \vdots \\ 0 \end{bmatrix}_{(2N,1)}$$

with  $\bar{v} = \sqrt{\bar{v}_x^2 + \bar{v}_y^2}$ . Instead, for low velocities:

$$A_v = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ -1 & & & & \\ & & \ddots & & \\ & & & -1 & \end{bmatrix}_{(4N,2N)} \quad b_v = \begin{bmatrix} v_{max} \\ \vdots \\ v_{max} \\ \Delta v_{max} \\ \Delta v_{max} \\ v_{max} \\ \vdots \\ v_{max} \\ \Delta v_{max} \\ \Delta v_{max} \end{bmatrix}_{(4N,1)}$$

It is important to note that the values corresponding to the last iteration impose more strict constraints, which can be expressed as:

$$\begin{cases} 0 \leq \frac{\bar{v}_x}{\bar{v}} v_x(k+N-1) + \frac{\bar{v}_y}{\bar{v}} v_y(k+N-1) \leq \Delta v_{max} & \text{for high velocities} \\ \begin{bmatrix} -\Delta v_{max} \\ -\Delta v_{max} \end{bmatrix} \leq \begin{bmatrix} v_x(k+N-1) \\ v_y(k+N-1) \end{bmatrix} \leq \begin{bmatrix} \Delta v_{max} \\ \Delta v_{max} \end{bmatrix} & \text{for low velocities} \end{cases}$$

This definition satisfies the requirement related to the fact that the vehicle should stop in proximity of the goal. The maximum value becomes then  $\Delta v_{max}$ , defined as:

$$\Delta v_{max} = \bar{a}_{max} \tau_{mpc}$$

where  $\bar{a}_{max}$  is the maximum average acceleration.

In conclusion, given the matrices defined above, it is possible to express the velocity constraints in the form:

$$A_v \mathcal{U}(k) \leq b_v$$

### 4.4.3 Velocity Variation Constraints

As previously introduced, the intrinsic limitations related to the system require the definition of constraints on the control variables. As described in [8], in order to guarantee safety and comfort on one side, and to take into account the actuation limit on the other, it is important to impose the saturation also with respect to the variation of velocity. This variation can be expressed by taking into account the maximum average acceleration.

In particular, it is possible to note that, since the MPC formulation is defined by means of discrete time instants, the velocity variation can be expressed as:

$$\Delta v(k) = v(k) - v(k - 1)$$

The constraint can be then defined as:

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

The threshold value  $\Delta v_{max}$  can be evaluated by considering the medium value of the acceleration between two consecutive iterations. The resulting time interval coincides with the sampling time of the MPC, therefore it is possible to write:

$$\bar{a} = \frac{\Delta v}{\tau_{mpc}}$$

In conclusion, the maximum variation of the velocity can be retrieved by considering the product between  $\tau_{mpc}$  and maximum value of the average acceleration:

$$\Delta v_{max} = \bar{a}_{max} \tau_{mpc}$$

The obtained result justifies the choice done in the previous section to define the maximum velocity in the last iteration of the algorithm.

In this particular case, the constraints will be imposed directly on the decoupled variables:

$$-\Delta v_{max} \leq v_x(k + i) - v_x(k + i - 1) \leq \Delta v_{max}$$

$$-\Delta v_{max} \leq v_y(k + i) - v_y(k + i - 1) \leq \Delta v_{max}$$

$$\forall i \in \{1, \dots, N - 1\}$$



a too much strict solution, which sometimes can bring to the unfeasibility of the problem, even if in the real case a compromise can be found. This is due to the fact that errors or uncertainties, that cannot be predicted by the controller, will drive the system outside the trust region.

Different solution were proposed in the past years, in general related to the introduction of additional heuristics, such as the removal of the constraints in certain conditions or the use of feasible solutions retrieved in the previous time instant, as highlighted in [11]. In this article, an optimal solution is proposed, by making a distinction between *hard* and *soft* constraints. The introduction of *slack variables* in the formulation will bring to a constraint relaxation and, consequently, to the solvability of the problem. The entire section will take into account the work done in [8] in order to formalize the above statement in a context related to an autonomous wheelchair.

#### 4.5.1 Cost Function

The slack variables can be considered as a part of the optimization variables, which will add particular features to the optimization problem. For this reason, it is fundamental to make the appropriate modifications to the formulation, compatibly with the new requirements. To this aim, a new cost function must be defined:

$$\bar{J}(k) = J(k) + \|u_{sl-p}(k)\|_{S_p}^2 + \|u_{sl-a}(k)\|_{S_a}^2$$

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

- $u_{sl-p}(k)$  is a vector of dimension  $n_{obs}$ , i.e. the number of obstacles considered in the formulation of position constraints, related to the slack on the state variables.
- $u_{sl-a}(k)$  represents the vector related to the slack on the velocity variation constraints along the two directions on the axes of the global reference frame.

By taking into account the above definitions, the weight matrices will be defined as:

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

The value of the parameters  $s_p$  and  $s_a$  will be chosen in order to allow the use of the slack variables only when there is a true need. In particular, by selecting a very high weight with respect to  $q$  and  $r$ , the controller will suggest values of the slack variables different from zero only if the feasibility of the problem will be compromised. By recalling the matrix form of the quadratic cost function, it is possible to define:

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

where

$$\bar{H} = \begin{bmatrix} H & & \\ & S_p & \\ & & S_a \end{bmatrix}_{(2N+n_{obs}+2, 2N+n_{obs}+2)}$$

$$\bar{f}^T = [f^T \quad 0_{(1, n_{obs})} \quad 0_{(1, 2)}]_{(1, 2N+n_{obs}+2)}$$

and

$$\bar{\mathcal{U}}(k) = \begin{bmatrix} \mathcal{U}(k) \\ u_{sl.p}(k) \\ u_{sl.a}(k) \end{bmatrix}_{(2N+n_{obs}+2, 1)}$$

#### 4.5.2 Soft Position Constraints

The slack variables related to the position are introduced to reduce the trust region of the state space, as shown in Figure 4.21, .

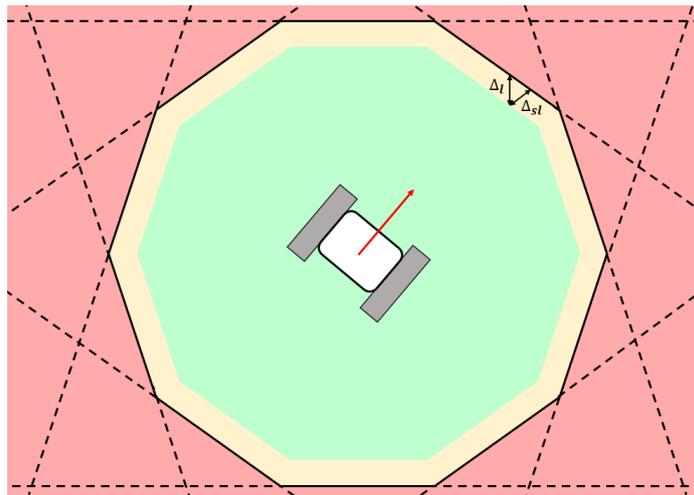


Figure 4.21: Trust Region with Hard and Soft Constrains

This choice has been done in relation to the fact that, if the problem admits a feasible solution, this is related to the position of point P defined by the Feedback Linearization. The reduction of the trust region will become useful to take into account the size of the vehicle during the execution of its motion.

The position constraints will be then defined as:

$$h_x^{(i)(j)}x(k+i) + h_y^{(i)(j)}y(k+i) \leq l^{(i)(j)} - \Delta l^{(j)}(1 - u_{sl-p}^{(j)}(k)) \quad (4.11)$$

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

where  $\Delta l^{(i)(j)} = \Delta l_{sl}^{(i)(j)} \sqrt{h_x^2 + h_y^2}$ . The coefficient  $\Delta l_{sl}$  represents the safety distance to be imposed in order to correctly include the size of the vehicle and, for this reason, must be chosen wisely.

By imposing to the slack variable to assume values between 0 and 1:

$$0 \leq u_{sl-p}^{(j)}(k) \leq 1$$

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

the condition (4.11) can be then written as:

$$\begin{cases} h_x^{(i)(j)}x(k+i) + h_y^{(i)(j)}y(k+i) \leq l^{(i)(j)}, & \text{if } u_{sl-p}^{(j)}(k) = 1 \\ h_x^{(i)(j)}x(k+i) + h_y^{(i)(j)}y(k+i) \leq l^{(i)(j)} - \Delta l^{(i)(j)}, & \text{if } u_{sl-p}^{(j)}(k) = 0 \end{cases}$$

It is now clear that the position slack variables are used to impose a constraint tightening, instead of a relaxation, compatibly with the safety requirements.

In order to extend the constraints to the overall prediction horizon, it is necessary to define:

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

where the vector different from zero occupies the  $j$ -th position.

The above matrix can be seen as an extraction one, such that:

$$u_{sl-p}^{(j)}(k) = E^{(j)} u_{sl-p}(k)$$

By considering the following variant of the position constraints:

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

where

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

it is possible to express condition (4.11) with respect to the set of control variables:

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

with

$$\bar{A}_{obs}^{(j)} = \begin{bmatrix} H_{obs}^{(j)} \mathcal{B} & E^{(j)} & 0_{(N+1,2)} \end{bmatrix}_{(N+1, 2N+n_{obs}+2)} \quad \bar{b}_{obs}^{(j)} = \bar{L}_{obs}^{(j)} - H_{obs}^{(j)} \mathcal{A} \xi(k)$$

It is now important to highlight the meaning of the matrix  $\bar{A}_{obs}^{(j)}$ . By recalling the definition of the vector of control variables:

$$\bar{\mathcal{U}}(k) = \begin{bmatrix} \mathcal{U}(k) \\ u_{sl,p}(k) \\ u_{sl,a}(k) \end{bmatrix}_{(2N+n_{obs}+2,1)}$$

it is immediately clear that:

- the first element  $H_{obs}^{(j)} \mathcal{B}$ , which represents the original position constraints, will be multiplied by  $\mathcal{U}(k)$ ;
- the second element  $E^{(j)}$ , which represents the contribution given by the position slack variable, will be multiplied by  $u_{sl,p}(k)$ ;
- the third element  $0_{(N+1,2)}$ , which represents a zeros matrix whose dimension is highlighted in the bracket, will nullify the contribution of the acceleration slack variable, which is useless in this case.

In conclusion, it is possible to define the entire set of position constraints as:

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

### 4.5.3 Soft Velocity Variation Constraints

In contrast with the position constraints, in the case of velocity variation, it is important to define a constraint relaxation. This is due to the fact that, even if limitations on the acceleration are important in order to guarantee safety and comfort, it may happen that the obstacle avoidance requirement will be compromised. In particular, if for example the obstacle moves towards the vehicle at a too much high velocity, the saturation on acceleration will inhibit the vehicle and, consequently, will cause collision. In that particular situation, the controller shall impose a higher acceleration, by preferring safety with respect to comfort. In order to formalize the above statement, it is possible to consider the following constraints:

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

the slack variables  $\Delta v_{sl_x}(k)$  and  $\Delta v_{sl_y}(k)$ , defined along the two axes directions, will impose an increment on the acceleration if the problem is not going to admit any feasible solution.

In order to guarantee safety during the execution, it is important however to impose the following constraints on the defined variables:

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

which will limit the increment of acceleration under the safety threshold  $\Delta v_{sl_{max}}$ .

As already discussed, the constraints must be expressed in the form:

$$\bar{A}_{\Delta v} \bar{U}(k) \leq b_{\Delta v}$$

where

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

With conclusions analogous to the soft position constraints implementation, the second matrix is defined as:

$$A_{\Delta sl} = \begin{bmatrix} 0 & \dots & 0 & \begin{bmatrix} -1 & 0 \end{bmatrix} \\ 0 & \dots & 0 & \begin{bmatrix} 0 & -1 \end{bmatrix} \\ \vdots & & & \vdots \\ 0 & \dots & 0 & \begin{bmatrix} -1 & 0 \end{bmatrix} \\ 0 & \dots & 0 & \begin{bmatrix} 0 & -1 \end{bmatrix} \\ \underbrace{\hspace{1.5cm}}_{u_{sl.p}(k)} & \underbrace{\hspace{1.5cm}}_{u_{sl.a}(k)} & & \end{bmatrix}_{(4N, n_{obs}+2)}$$

#### 4.5.4 Slack Variables Constraints

As previously described, the introduction of particular characteristics in the implementation of the soft constraints, forced the introduction of limits in the definition of the slack variables. In particular, the constraints on the position slack:

$$\begin{aligned} 0 &\leq u_{sl,p}^{(j)}(k) \leq 1 \\ \forall j &\in \{0, \dots, n_{obs}\} \end{aligned}$$

where introduced to limit the value of the variables between 0 and 1, in order to avoid to extend the trust region too much, with the risk to include also obstacles in the free space.

Furthermore, the constraints related to the acceleration slack

$$\begin{aligned} 0 &\leq \Delta v_{sl,x}(k) \leq \Delta v_{sl,max} \\ 0 &\leq \Delta v_{sl,y}(k) \leq \Delta v_{sl,max} \end{aligned}$$

force the increment to assume values that will guarantee safety and the satisfaction of the actuation limitations. By extending the previous statements along the entire prediction horizon:

$$A_{sl,p} = \begin{bmatrix} 0 & \dots & 0 & 1 & \dots & 0 & 0 & 0 \\ \vdots & & \vdots & & \ddots & & \vdots & \vdots \\ 0 & & 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & & 0 & -1 & \dots & 0 & 0 & 0 \\ \vdots & & \vdots & & \ddots & & \vdots & \vdots \\ 0 & \dots & 0 & 0 & \dots & -1 & 0 & 0 \end{bmatrix}_{(2n_{obs}, 2N+n_{obs}+2)} \quad b_{sl,p} = \begin{bmatrix} 1 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}_{(2n_{obs}, 1)}$$

and

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

it is possible to obtain the classical representation:

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

with

$$A_{sl} = \begin{bmatrix} A_{sl,p} \\ A_{sl,a} \end{bmatrix} \quad b_{sl} = \begin{bmatrix} b_{sl,p} \\ b_{sl,a} \end{bmatrix}$$

## 4.6 Final Pose Control

As already discussed, in order to obtain a linear decoupled system, the Feedback Linearization has been introduced. The resulting system can be interpreted as the representation of the motion of a particle in a plane, as highlighted in Figure 4.22. The controller will receive a trajectory, computed

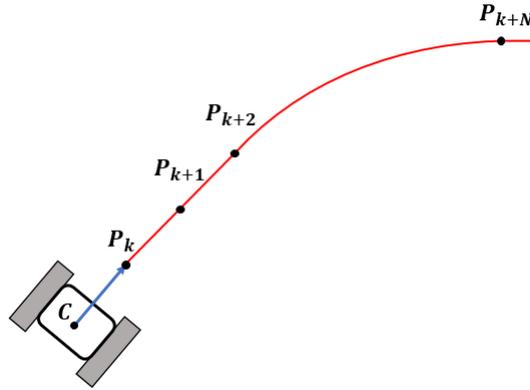


Figure 4.22: Trajectory of  $P$

by the global planner, which will be segmented. Each point will be then used in order to define a series of position references, that the vehicle will try to reach. In this scenario, the orientation of the vehicle loses importance as a state variable, but it will be used as an additional information, in order to retrieve the transformation matrix:

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

However, as it will be described in the next chapter, the global planner will impose to the local planner also an orientation reference. In the case in which the orientation goal is different from the one related to the tangent of the trajectory, a further control, focused on the final pose of the vehicle, is needed. As already introduced, the MPC has been designed for a regulation purpose. For this reason, it is possible to consider the position control and the final pose one as two independent problems. To this aim, it is possible to define a *Control Hierarchy*, as described in Figure 4.27 at the end of the chapter.

As it is possible to note, if the orientation control is required, this shall act on the angular velocity around the odometric centre  $C$  of the vehicle, once the position goal has been reached.

However, the position reference is expressed with respect to the point  $P$  obtained through the *Feedback Linearization*. In this scenario, the vehicle will stop at a distance  $\varepsilon$  from the goal. Therefore, as highlighted in [8], the

first operation to be done is to transform the reference for  $P$  into a goal for the odometric centre  $C$ .

The reference for the  $C$  can be expressed as:

$$\begin{cases} x_{C_{ref}} = x_{P_{ref}} + \varepsilon \cos \theta_{ref} \\ y_{C_{ref}} = y_{P_{ref}} + \varepsilon \sin \theta_{ref} \end{cases}$$

where  $(x_{P_{ref}}, y_{P_{ref}})$ ,  $(x_{C_{ref}}, y_{C_{ref}})$  and  $\theta_{ref}$  are the references of point  $P$ , of the odometric centre  $C$  and of the orientation of the vehicle respectively, while  $\varepsilon$  is the parameter related to the Feedback Linearization.

In order to make possible for the odometric centre to converge to the given goal, it is necessary to introduce into the MPC formulation a new reference for  $P$ , as shown in Figure 4.23.

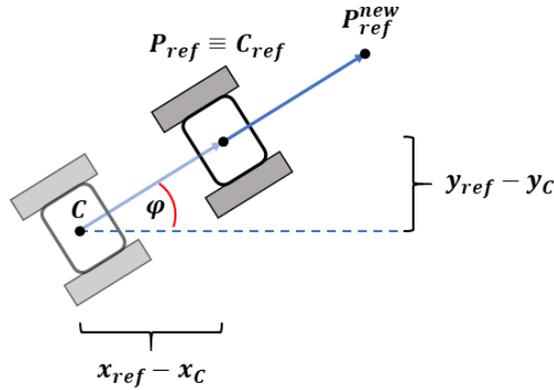


Figure 4.23: New Reference

The new reference is therefore:

$$P_{ref}^{new} : \begin{cases} x_{P_{ref}^{new}} = x_{C_{ref}} - \varepsilon \cos \varphi \\ y_{P_{ref}^{new}} = y_{C_{ref}} - \varepsilon \sin \varphi \end{cases}$$

where  $\varphi = \arctan\left(\frac{y_{C_{ref}} - y_c}{x_{C_{ref}} - x_c}\right)$  is defined in this way in order to keep the orientation unchanged in this step.

Once the odometric centre has reached its goal, it is possible to deactivate the MPC, in order to apply the orientation control, acting on the angular velocity. By recalling that the representation of the orientation as a state variable can be written as:

$$\dot{\theta}(t) = \omega(t)$$

it is possible to see how the system is described by an integrator. The orientation control scheme is then really simple, as shown in Figure 4.24.

The block  $G(s) = \frac{\mu}{s} e^{-\tau s}$  represents the system transfer function, in which the actuation delay  $\tau$  is considered.

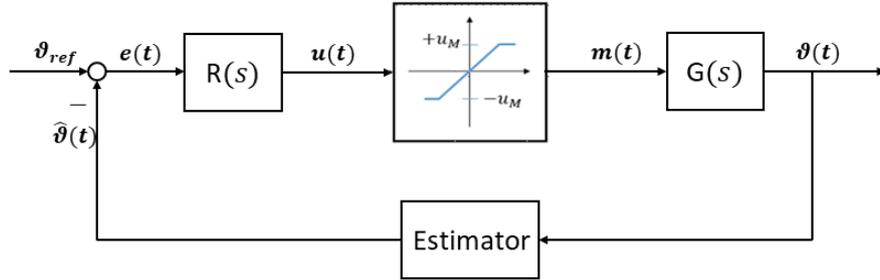


Figure 4.24: Orientation Control Scheme

By looking at the system transfer function, it is clear that a simple proportional action can be used to define the regulator transfer function:

$$R(s) = K_{\theta}$$

By considering the closed-loop function:

$$L(s) = R(s)G(s) = \frac{K_{\theta}\mu}{s}e^{-\tau s}$$

and the Bode Diagram of its modulus, represented in Figure 4.25,

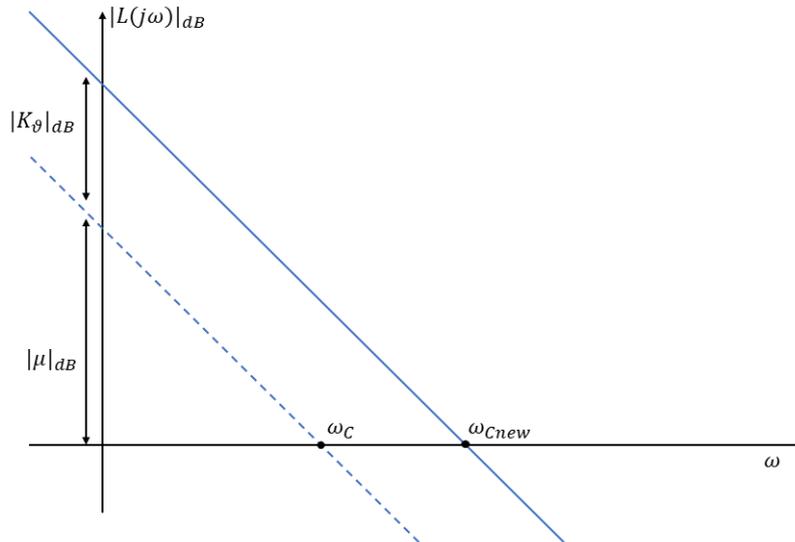


Figure 4.25: Modulus Bode Diagram

it is possible to select the gain of the regulator such that:

$$K_{\theta} = \frac{\omega_{Cnew}}{\mu}$$

The contribution of  $e^{-\tau s}$  will be considered only as a delay in the phase:

$$\varphi_\tau = -\omega_{C_{new}}\tau \frac{180}{\pi}$$

In order to take into account the safety and comfort requirements, a limitation on the control variable is needed. In this particular case it is enough to insert a saturation block, as already highlighted in the control scheme, defined as:

$$m(t) = \begin{cases} -u_M, & u(t) < -u_M \\ u(t), & |u(t)| \leq u_M \\ u_M, & u(t) > u_M \end{cases}$$

where  $u_M = \omega_{max}$  expresses the maximum angular velocity to which the system shall be subjected.

In conclusion, the signal  $e(t)$  must be designed in order to keep the error inside the interval  $(-\pi, \pi]$ :

$$e(t) = \begin{cases} |\theta_{ref} - \hat{\theta}| & \text{if } |\theta_{ref} - \hat{\theta}| \leq \pi \\ |\theta_{ref} - \hat{\theta}| - 2\pi & \text{otherwise} \end{cases}$$

where  $\hat{\theta}$  is the estimated orientation of the vehicle.

Further information about the above control approach can be found in [17].

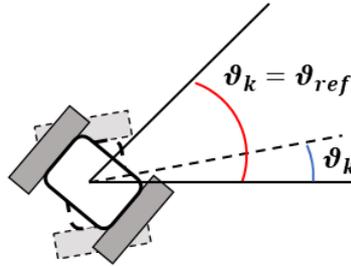


Figure 4.26: Controlled System

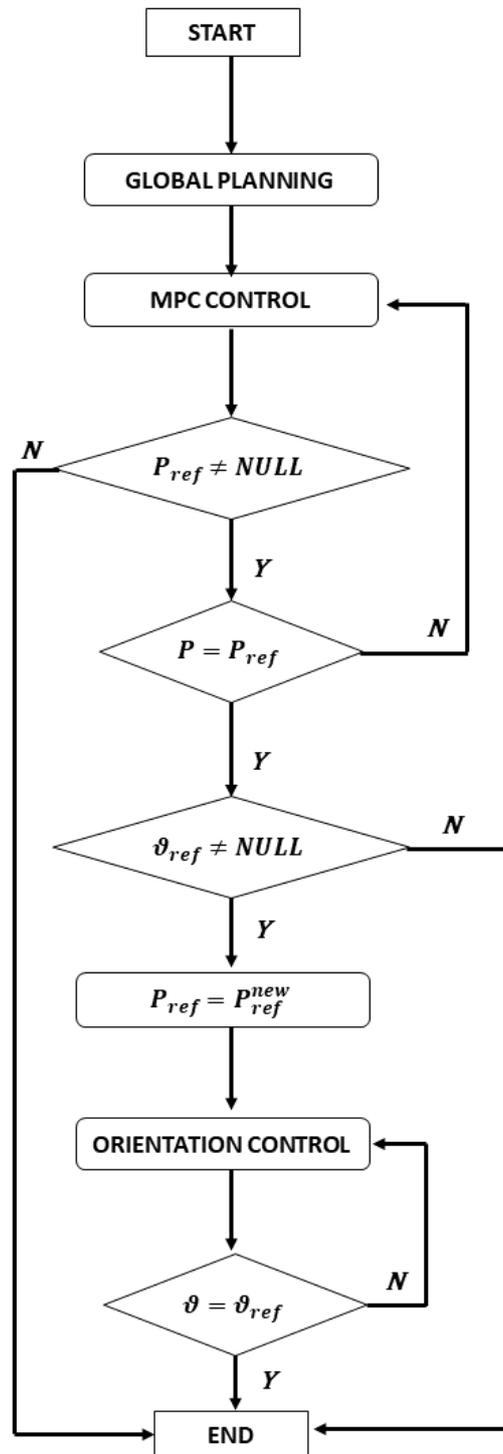


Figure 4.27: Control Hierarchy



## Chapter 5

# RRT\* Motion Primitives

This chapter focuses on the description of the approach adopted in this work for global planning. In the first part, an introduction of the kinodynamic motion planning problem will be presented. The second section will introduce the RRT approach and its variants. Lastly, the algorithm formulated in [5] will be used in the particular context of a crowded environment.

### 5.1 Motion Planning Problem

In the field of robotics, motion planning is fundamental to define the behaviour of a robot inside a given environment. In this context, a key role is covered by the concept of *configuration*. This particular term is used to express the set of positions and orientations that completely identify the robot inside the space. For sake of clarity, the unicycle model will be taken into account:

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

where the variables  $x(t)$  and  $y(t)$  are the positions along the global reference axes of the system, while  $\theta(t)$  is the orientation of the vehicle. The longitudinal velocity  $v(t)$  and the angular speed  $\omega(t)$ , which describes the rotation around the axis perpendicular with respect to the motion plane, allow to define the vector of control variables  $u(t) = [v(t) \ \omega(t)]$ . In order to characterize a mobile robot by means of this particular model, it is possible to introduce the vector of generalized coordinates:

$$q(t) = [x(t) \ y(t) \ \theta(t)]^T$$

where  $q(t)$  identifies the robot configuration.

The motion planning problem can be now defined as the research of a

motion characterized by a starting configuration  $q_s = [x_s \ y_s \ \theta_s]^T$  and a goal one  $q_g = [x_g \ y_g \ \theta_g]^T$ , as shown in Figure 5.1.

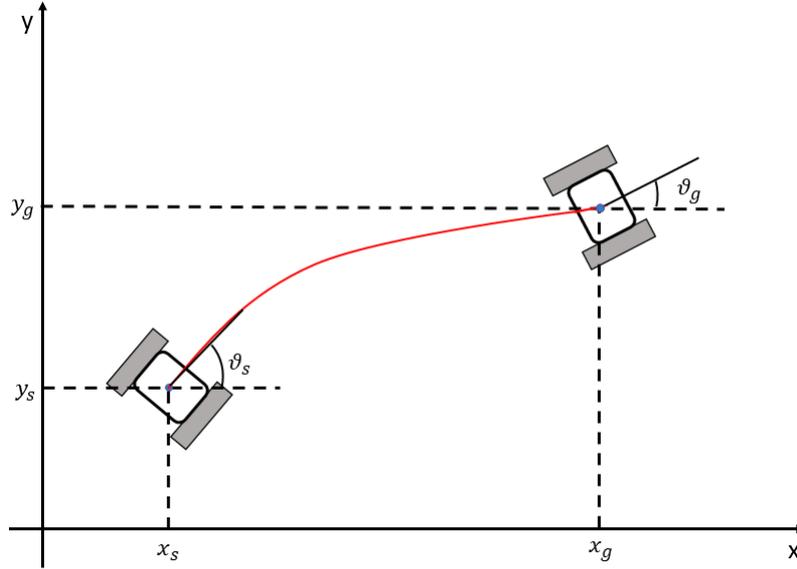


Figure 5.1: Motion Planning Problem

As it is clear, a motion is generated through the definition of a trajectory connecting two points. Therefore, the behaviour of a robot in the space is highly dependent from the way in which the trajectory is defined. In addition, in order to obtain a desirable motion, the planning problem shall satisfy different constraints, which can be distinguished into:

- *Hard bounds*, related to the limitations on velocity, acceleration and torques.
- *Differential constraints*, related to the dynamics of the system, described as:

$$\dot{s}(t) = f(s(t), u(t)) \quad (5.1)$$

where  $f$  is a suitable function, while  $s(t)$  and  $u(t)$  are the state and control variables respectively;

- *Non holonomic*, which represent every constraint that cannot be integrated in order to produce direct limitations on the state.

The above constraints are parts of the class of *Kinodynamic Constraints*. Early planning methods did not take into account the above constraints and, for this reason, the system could not always execute the motion suggested by the problem solution. To this aim, [3] presented the *Kinodynamic Motion Planning*.

### 5.1.1 Kinodynamic Motion Planning

In order to formalize the problem, some basic concepts are required:

- *State space* - Given a set of configurations of a robot in the space  $Q \subset \mathbb{R}^{n_q}$ , where  $n_q$  identifies the number of independent variables used to describe a single configuration, it is possible to define the state of a robot as the vector  $s = (q, \dot{q})$  with  $q \in Q$ . In general, the state of a system is identified as the configuration of the robot in the space and its time derivatives. Given a state  $s$  it is possible to define a function  $\lambda : S \rightarrow Q$  which maps the state  $s \in S$  to the equivalent configuration  $q \in Q$ , such that  $q = \lambda(s)$ . It is then possible to define a subset of the state space  $S$ , denoted as  $S_{free}$ , such that:

$$S_{free} := \{s \in S \mid \lambda(s) \in Q_{free}\}$$

where  $Q_{free} = Q/Q_{occ}$  denotes the free space in the robot framework, which corresponds to the set of collision-free configurations, while  $Q_{occ}$  defines the set of configurations which bring the vehicle to the space occupied by obstacles. It is then clear that  $S_{free}$  identifies the free-space in the state space and, consequently, it is possible to introduce in the specified set also the constraints related to the state. Given a set of inequalities in the form  $h(s) \leq 0$ , the previously defined set becomes:

$$S_{free} := \{s \in S \mid \lambda(s) \in Q_{free} \wedge h(s) \leq 0\}$$

Moreover, from the set of free configurations  $Q_{free}$  it is possible to extract the so-called *Goal Configuration Space*  $Q_{goal}$ , defined as the subset of collision-free configurations that the robot must reach at the end of the motion. Consequently, the *Goal Region*  $S_{goal} \subset S_{free}$  can be defined as:

$$S_{goal} := \{s \in S \mid \lambda(s) \in Q_{goal}\}$$

- *Trajectory* - A trajectory  $\sigma(t) : [0, T] \rightarrow S$  is a particular function which identifies a path, once a timing law between 0 and  $T$  has been assigned. It can be retrieved through integration of Equation 5.1, defining an initial condition  $\sigma(0) = s_0$  and by introducing a control function  $u : [0, T] \rightarrow U$ , where  $U$  takes the name of *Control Space* and represents the set of all the available control actions.

Aim of the Kinodynamic Trajectory Planning is to find a feasible trajectory  $\sigma(t) : [0, T] \rightarrow S$  such that:

- it starts from an initial point  $\sigma(0) = s_0$ ;
- it reaches the goal region within  $T$ , namely  $\sigma(T) \in S_{goal}$ ;

- it satisfies the control variables limitations, such that  $u(t) \in U_{limit}$ , where  $U_{limit}$  is the set of control variables satisfying the related bounds;
- it satisfies the kinodynamic constraints;
- it is collision-free, i.e.  $\sigma(t) \in S_{free}, \forall t \in [0, T]$ .

## 5.2 RRT

As described in [6], in the field of kinodynamic motion planning, a large variety of approaches have been introduced in the years. A fundamental impact is given by RRT (Rapidly-Exploring Random Tree) approach ([22]) and its variants, belonging to the class of *Sampling-Based Methods*, which will be used in this thesis. The RRT approach is summarized in Algorithm 1.

---

### Algorithm 1 RRT

---

```

1:  $N \leftarrow \{s_0\}; E \leftarrow 0; i \leftarrow 0; N_{goal} \leftarrow 0$ 
2: for  $i < n_{iter}$  do
3:    $s_{rand} \leftarrow \mathbf{Sample}(S_{free});$ 
4:    $s_{near} \leftarrow \mathbf{NearestNode}(N, s_{rand});$ 
5:    $s_{new}, \sigma_{new} \leftarrow \mathbf{Steer}(s_{near}, s_{rand});$ 
6:   if  $\mathbf{isCollisionFree}(\sigma_{new})$  then
7:      $N \leftarrow N \cup s_{new}, E \leftarrow E \cup \{s_{near}, s_{new}\}$ 
8:     if  $N_{goal} \neq 0$  then
9:       return  $\mathbf{BestTraj}(N_{goal})$ 
10:    end if
11:  end if
12:   $i \leftarrow i + 1$ 
13: end for

```

---

RRT and, more in general, the majority of sampling-based algorithms, relies on the construction of a graph, or a tree, representing a roadmap of feasible trajectories which can be followed by a robot in order to reach a goal state  $s_g$  from a starting one  $s_0$ . Each node of the graph represents the state of the dynamic system, while the connecting edges correspond to the trajectories built by starting from a node  $s_1 \in S$  and ending in a node  $s_2 \in S$ . A graph structure can be then formalized as:

$$G = (N, E)$$

where  $N$  represents the set of nodes, while  $E$  expresses the set of the edges belonging to the graph. Compatibly with the above definition it is immediately clear that:

$$e = \{s_1, s_2\} \in E$$

where  $s_1, s_2 \in S$  are two states of the system, and  $e$  is the edge identifying the trajectory from  $s_1$  to  $s_2$ , which can be expressed as  $\sigma_{s_1 \rightarrow s_2}(t)$ . If a goal region has been defined, then the sets of nodes belonging to it can be formalized as:

$$N_{goal} := \{s \in N \mid s \in S_{goal}\}$$

The construction of the tree structure starts with the function **Sample**( $S_{free}$ ), with which a random sample  $s_{rand}$ , representing the state of the robot and assumed to be independent and identically distributed, is selected at each step from the continuous collision-free state space  $S_{free}$  with a uniform probability distribution. From the set of nodes  $N$  already belonging to the tree under construction, the procedure **NearestNode**( $N, s_{rand}$ ) finds the nearest node  $s_{near}$  with respect to  $s_{rand}$ . The concept of distance plays a fundamental role in order to include the idea of nearness. For this reason, the introduction of a distance metric must be required. In general, if the path planning is performed inside an euclidean space, the metric can be simply defined by choosing the euclidean distance:

$$d(s, s_{rand}) = \|s - s_{rand}\|$$

However, for more complex state spaces, as for example the ones defined by the higher order derivatives of the robot configurations, the previously introduced metric becomes meaningful. For this reason, a more general solution relies on the introduction of the idea of cost of the trajectory connecting two states. An example of cost can be defined as the time needed to perform a given trajectory. However, it is important to note that the cost cannot be seen as a true metric since, unlike the euclidean distance, it cannot be symmetric with respect to the nodes. Once  $s_{near}$  is found, the **Steer**( $s_{near}, s_{rand}$ ) function generates a third sample, called  $s_{new}$ , at a pre-defined distance  $\delta$  from  $s_{near}$ . Then, through a procedure called *steering* (whence the name of the function), a trajectory  $\sigma_{new}$  is generated in order to connect  $s_{near}$  with  $s_{new}$ . In order to expand the tree, it will be checked if the trajectory is collision-free and, in case the condition is true,  $s_{new}$  will be incorporated to the set of nodes  $N$ , while  $\sigma_{new}$  will be added to the set of the edges  $E$ . Reached the maximum number of iterations  $n_{iter}$ , if the tree can be connected to the goal  $s_g$  and to the initial state  $s_0$  in some way, the best trajectory, computed by the function **BestTraj**( $N_{goal}$ ), will be used to reach  $s_g$  from  $s_0$ . Figure 5.2 shows the main steps of the described procedure.

The main property of RRT is related to its *Probabilistic Completeness*, based on the fact that the probability to find a feasible solution, i.e. if the computed graph contains a node in the goal region  $S_{goal}$ , will tend to one as the computation time tends to infinity. For this reason it is fundamental to set an upper bound to the number of iterations of the algorithm, in order to make it stop if a feasible solution is still not found.

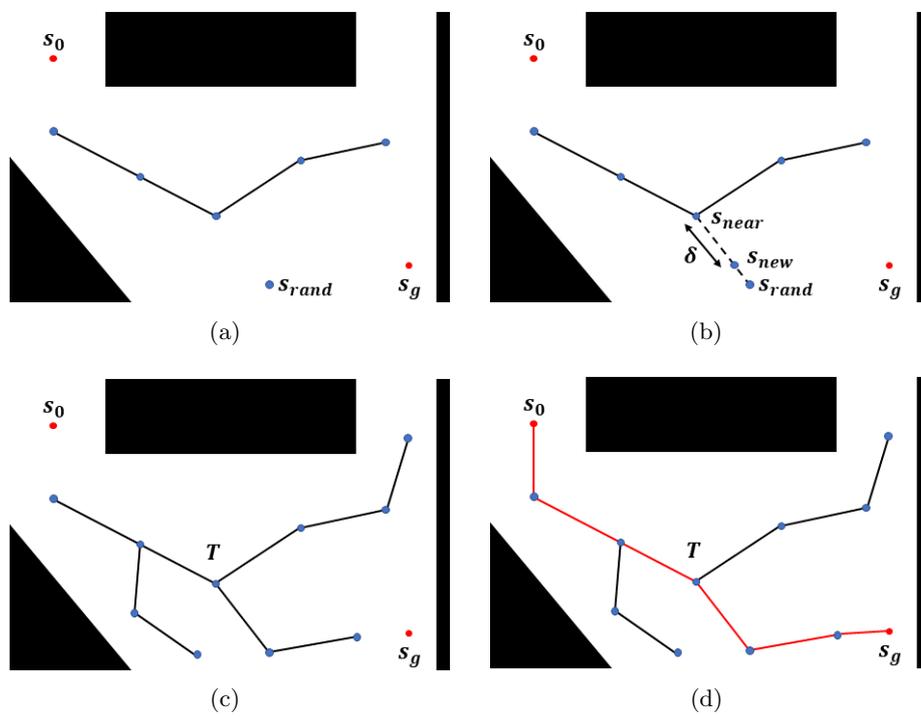


Figure 5.2: RRT Procedure: 5.2(a) Sample Extraction 5.2(b) Collision Checking 5.2(c) Maximum Iteration Reached 5.2(d) Best Path

### 5.3 RRT\*

The RRT\* variant, described in [21], is introduced in order to ensure the asymptotic optimality of the retrieved solution. The property of optimality is strictly related to the idea of cost, and in particular to the concept of *Optimal Steering*.

Given a function which associates a cost to each feasible trajectory  $\sigma(t)$ , defined as  $Cost(\sigma)$ , the optimal steering problem is related to the computation of an optimal trajectory  $\sigma^*(t) : [0 : T^*] \rightarrow S$ , based on a set of states  $s^*(t) : [0 : T^*] \rightarrow S$  and a set of control variables  $u^*(t) : [0 : T^*] \rightarrow U$ , which satisfies the following properties:

- it starts from  $\sigma^*(0) = s_0$ , where  $s_0$  is the initial state;
- it reaches the final state in  $T^*$ , such that  $\sigma^*(T^*) = s_f$ ;
- it minimizes the function  $Cost(\sigma^*)$ ;
- it satisfies the constraints related to the system.

By taking into account the previous statement, it is clear that, in order to guarantee optimality, each new node must be connected by ensuring the minimization of the cost function.

The property of asymptotic optimality ensures that the probability of finding a path that reaches a goal region, composed by a set of optimal trajectories, tends to one as the number of iterations of the algorithm tends to infinity:

$$P\left(\lim_{i \rightarrow \infty} Cost(\sigma_{TOT}^i) = Cost(\sigma_{TOT}^*)\right) = 1$$

where  $\sigma_{TOT}^i = \{\sigma_1, \dots, \sigma_{goal}\}$  is the path retrieved at the  $i$ -th iteration, while  $\sigma_{TOT}^* = \{\sigma_1^*, \dots, \sigma_{goal}^*\}$  is the optimal solution of the path planning problem. The RRT\* formulation is summarized in Algorithm 2.

As in the RRT procedure, a random sample  $s_{rand}$  is selected from the collision-free state space. However, instead of selecting the nearest node, the function **NearNodes**( $s_{rand}$ ) returns the set of nearby nodes  $N_{near}$ , with respect to  $s_{rand}$ , defined as:

$$N_{near} := \{s \in N \mid dist(s_{rand}, s) \leq \bar{d}\}$$

where  $\bar{d}$  is a predefined distance. However, it is important to note, as previously introduced, that if the state space is not euclidean, then the distance function is not symmetric, i.e.  $dist(s, s_{rand}) \neq dist(s_{rand}, s)$ . The main procedure, which goes under the name of **Extend**, is highlighted in Algorithm 3.

The function **Extend** works in a very simple way. For each sample  $s_{near}$  from the set of nearby nodes with respect to  $s_{rand}$  the trajectory  $\sigma_{near}$

**Algorithm 2** RRT\*

---

```

1:  $N_i \leftarrow \{s_0\}; E_i \leftarrow 0; i \leftarrow 0;$ 
2: for  $i < n_{iter}$  do
3:    $s_{rand} \leftarrow \mathbf{Sample}(S_{free})$ 
4:    $N_{near} \leftarrow \mathbf{NearNodes}(s_{rand})$ 
5:    $s_{best} \leftarrow \mathbf{Extend}(N_{near}, s_{rand})$ 
6:   if  $s_{best} \neq 0$  then
7:      $N_{i+1} \leftarrow N_i \cup \{s_{rand}\}$ 
8:      $E_{i+1} \leftarrow E_i \cup \{s_{best}, s_{rand}\}$ 
9:      $i \leftarrow i + 1$ 
10:     $E_{i+1} \leftarrow \mathbf{Rewire}(N_{i+1}, E_{i+1}, s_{rand}, N_{near})$ 
11:   end if
12: end for
13: return  $N_{n_{iter}}, E_{n_{iter}}$ 

```

---

**Algorithm 3**  $\mathbf{Extend}(N_{near}, s_{rand})$ 


---

```

1:  $s_{best} \leftarrow 0; e_{best} \leftarrow 0; c_{best} \leftarrow \infty$ 
2: for  $\forall s_{near} \in N_{near}$  do
3:    $\sigma_{near}, Cost(\sigma_{near}) \leftarrow \mathbf{Steer}(s_{near}, s_{rand});$ 
4:   if  $Cost(\sigma_{near}) < c_{best}$  then
5:     if  $\mathbf{isCollisionFree}(\sigma_{near})$  then
6:        $s_{best} \leftarrow s_{near};$ 
7:        $e_{best} \leftarrow \sigma_{near};$ 
8:        $c_{best} \leftarrow Cost(\sigma_{near});$ 
9:     end if
10:  end if
11: end for
12: return  $s_{best}$ 

```

---

connecting them and the cost associated to the given trajectory  $\mathbf{Cost}(\sigma_{near})$  are computed. If the trajectory is collision-free and, in addition, it is the one with the minimum cost, then the state  $s_{near}$  associated to the trajectory will be returned and added to the tree.

The last procedure of the algorithm, called **Rewire** checks if, once  $s_{best}$  is added to the tree, there are nodes in  $N_{near}$  which can be reached from  $s_{best}$  with lower cost and through a collision-free trajectory. If the condition is satisfied, the tree is "rewired" and the new edge is added, by discarding the previous one.

The asymptotically optimality property is satisfied through the introduction of the **Rewire** function and by a particular selection of the set of nearby nodes. The choice is related to the definition of a hypersphere centred in  $s_{rand}$ , whose extension depends on the cardinality of the tree.

Therefore the nearby nodes that will be chosen are the ones inside the sphere volume. If the system is holonomic and defined in an euclidean space, the metrical distance becomes a d-dimensional hypersphere, whose radius is defined as:

$$\gamma_{sphere} = \gamma_{RRT^*} \left( \frac{\log(n)}{n} \right)^{\frac{1}{d}}$$

where  $n = |N|$  represents the cardinality of the tree, while  $\gamma_{RRT^*}$  is a constant whose value is selected as:

$$\gamma_{RRT^*} > 2 \left( 1 + \frac{1}{d} \right)^{\frac{1}{d}} \left( \left( \frac{\mu(Q_{free})}{\zeta_d} \right)^{\frac{1}{d}} \right)$$

where  $\mu(Q_{free})$  and  $\zeta_d$  are respectively the volume of the free configuration space and the volume of the sphere, as described in [7].

## 5.4 RRT\* Motion Primitives

The path planning approach adopted in this thesis relies on a novel variant of RRT\* proposed in [4] which combines the sampling-based motion planning with a search-based one. In particular, it relies on a discretized state space  $S^\Delta$  and on the definition of a database of Motion Primitives.

### 5.4.1 Sampling-Based planning with Motion Primitives

As highlighted in [4] the introduction of a database of Motion Primitives, highly reduces the on-line computational load of sampling-based approaches. This is due to the fact that, each time a steering procedure is called by the RRT\* algorithm, in order to compute the best trajectory connecting two states, an optimization problem must be solved on-line. For this reason, the steering procedure is substituted by the extraction of an already computed feasible motion from the database.

The Motion Primitives database is built by considering a gridded state space  $S^\Delta$ . For each couple of states  $s_0, s_f \in S^\Delta$  a TPBVP (Two-Point Boundary-Value Problem) is defined and solved, retrieving a motion primitive which will be stored inside the database. The optimization problem for the case study of this thesis can be defined by taking into account the unicycle model. For each pair of states  $s_0 = (\bar{x}_0, \bar{y}_0, \bar{\theta}_0, \bar{v}_0)$  and  $s_f = (\bar{x}_f, \bar{y}_f, \bar{\theta}_f, \bar{v}_f)$ , the TPBVP takes the form of:

$$\begin{aligned}
& \min_{a(t), \omega(t), \tau} \int_0^\tau (1 + r_a a^2(t) + r_\omega \omega^2(t)) dt \\
& \text{subject to } \dot{x}(t) = v(t) \cos \theta(t) \\
& \quad \dot{y}(t) = v(t) \sin \theta(t) \\
& \quad \dot{\theta}(t) = \omega(t) \\
& \quad \dot{v}(t) = a(t) \\
& \quad a(t) \in [0, a_{max}] \quad t \in [0, \tau] \\
& \quad \omega(t) \in [0, \omega_{max}] \quad t \in [0, \tau] \\
& \\
& \quad x(0) = \bar{x}_0 \\
& \quad y(0) = \bar{y}_0 \\
& \quad \theta(0) = \bar{\theta}_0 \\
& \quad v(0) = \bar{v}_0 \\
& \quad x(\tau) = \bar{x}_f \\
& \quad y(\tau) = \bar{y}_f \\
& \quad \theta(\tau) = \bar{\theta}_f \\
& \quad v(\tau) = \bar{v}_f
\end{aligned}$$

where  $a(t)$  and  $\omega(t)$  are the control variables, while  $r_a$  and  $r_\omega$  are suitable weight constants.

It is important to note how the cost function is designed in order to minimize the execution time of each trajectory, ensuring the satisfaction of the system limitations. Figure 5.3, taken from [4], shows an example of a motion primitives database, computed by taking into account the unicycle model.

With the right assumption, it is possible to keep the size of the database small, despite a high dimensional configuration space is considered. This result can be obtained by taking into account the *translation invariance property*, which states that, given the position  $\pi$  of a vehicle, expressed with respect to a given reference system and its state evolution  $\sigma(t)$ , retrieved by applying a given input  $u(t)$ , if the system dynamics of the vehicle and its initial state  $s^{new}$  are expressed with a new coordinate system, then the state evolution  $\sigma^{new}(t)$ , retrieved by applying  $u(t)$ , is the translated version of  $\sigma(t)$ . This particular property allows to centre the initial position, used to build the database, at the origin of the reference system, by simply setting:

$$\pi_0 = (x_0, y_0, \theta_0) = 0_{(1,3)}$$

where  $\pi$  expresses the position components of the state. Moreover, it is possible to define a *Bounding Box* which takes into account all the grid

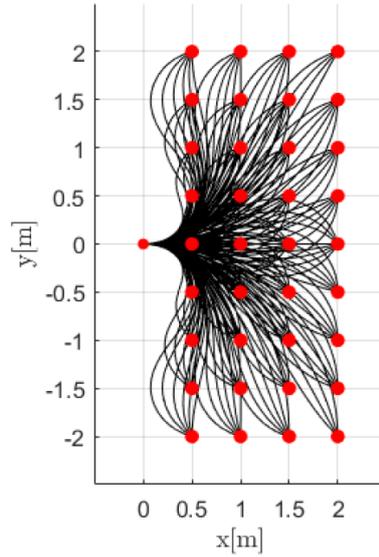


Figure 5.3: "A subset of the motion primitives computed for a 3D search space  $(x; y; \theta)$ . Red dots correspond to the Cartesian coordinates  $(x; y; \theta)$  and black lines represent the resulting trajectories for different final orientations", pg.30, *Optimal Kinodynamic Planning For Autonomous Vehicles*, B.Sakcak.

points used to compute the trajectories connecting each pair of initial and final states inside the database. In particular,  $\mathbf{BoundingBox}(\pi_0)$  represents the set of boundary values centred in  $\pi_0 = (x_0, y_0, \theta_0)$ .

### 5.4.2 Motion Planning

RRT\* Motion Primitives approach follows the main procedure of sampling-based algorithms in the definition of a tree  $T = (N, E)$ , with the main difference that the states  $s \in N$  are sampled from a discretized state space  $S^\Delta$ , while  $e \in E$  are the optimal trajectories extracted from the database of motion primitives. The procedure is described in Algorithm 4. As it is shown, the approach presented is defined by means of four main steps.

#### Random Sampling

The function  $\mathbf{Sample}(S_{free}^\Delta)$  selects a random sample  $s_{rand}$  from the free State Space  $S_{free}^\Delta$ , defined as:

$$S_{free}^\Delta := \{s \in S^\Delta \mid \lambda(s) \in Q_{free} \wedge h(s) \leq 0\}$$

where  $\lambda(s) = q$  is the function which maps a state into a configuration, while  $h(s)$  are the algebraic relationships defining the limitations on the state variables.

**Algorithm 4** RRT\* Motion Primitives

---

```

1:  $N_i \leftarrow \{s_0\}; E_i \leftarrow 0; i \leftarrow 0;$ 
2: for  $i < n_{iter}$  do
3:    $s_{rand} \leftarrow \mathbf{Sample}(S_{free}^\Delta)$ 
4:    $N_{near} \leftarrow \mathbf{NearNodesToCome}(s_{rand})$ 
5:    $s_{best}, e_{best} \leftarrow \mathbf{FindBestParent}(N_{near}, s_{rand})$ 
6:   if  $s_{rand} \in N$  then
7:      $(N_{i+1}, E_{i+1}) \leftarrow \mathbf{UpdateExisting}(s_{best}, e_{best}, s_{rand})$ 
8:   else if  $s_{rand} \notin N \wedge s_{best} \neq 0$  then
9:      $(N_{i+1}, E_{i+1}) \leftarrow \mathbf{UpdateNew}(s_{best}, e_{best}, s_{rand})$ 
10:  end if
11:   $i \leftarrow i + 1$ 
12: end for
13: return  $\mathbf{BestTraj}(N_{goal})$ 

```

---

It is important to note that, in contrast with the other search-based approaches, this random sampling procedure extracts  $s_{rand}$  from a predefined grid.

**Near Nodes**

It is fundamental to define a distance metric in order to construct a neighbourhood of nodes. To this aim, it is convenient to rely on the database of motion primitives, in particular on the cost of the optimal trajectories. Therefore, the distance metric becomes:

$$dist(s_1, s_2) = Cost(\sigma_{s_1 \rightarrow s_2})$$

The main drawback of this choice is related to the fact that the cost function is not symmetric:

$$Cost(\sigma_{s_1 \rightarrow s_2}) \neq Cost(\sigma_{s_2 \rightarrow s_1})$$

For this reason, two different procedures must be introduced. In particular, Algorithm 5 describes the **NearNodesToCome** function, which is used when a trajectory reaching  $s_{rand}$  must be defined.

To this aim, by recalling the procedure introduced for the RRT\* in order to guarantee the asymptotic optimality, the reachable nodes can be defined as:

$$N_{ReachToCome} = \{s \in N : dist(s, s_{rand}) \leq \bar{d}(n)\}$$

where  $\bar{d}(n) = \gamma \left( \frac{\log(n)}{n} \right)$ , while  $\gamma$  is a constant selected in a way such that the Euclidean ball  $\gamma_l^d \left( \frac{\log(n)}{n} \right)$ , function of the cardinality of the tree  $n = |N|$ , is contained within  $N_{ReachToCome}$ .

However, the above region may include reachable states that cannot

**Algorithm 5**  $\text{NearNodesToCome}(s_{rand})$ 


---

```

1:  $N_{near} \leftarrow 0$ ;
2:  $\pi_{rand} \leftarrow \text{GetPosition}(s_{rand})$ 
3: for  $\forall s \in N$  do
4:   if  $s \in \text{BoundingBox}(\pi_{rand})$  then
5:      $\sigma_{s \rightarrow s_{rand}} \leftarrow \text{FindTrajectory}(s, s_{rand})$ 
6:     if  $\text{Cost}(\sigma_{s \rightarrow s_{rand}}) \leq \bar{d}(n)$  then
7:        $N_{near} \leftarrow N_{near} \cup \{s\}$ 
8:     end if
9:   end if
10: end for
11: return  $N_{near}$ 

```

---

be mapped inside the database of motion primitives, making impossible to apply the search-based procedure. For this reason, the set of nearby nodes, as shown in Figure 5.4, will be defined as:

$$N_{NearToCome} = N_{ReachToCome} \cap \text{BoundingBox}(\pi_{rand})$$

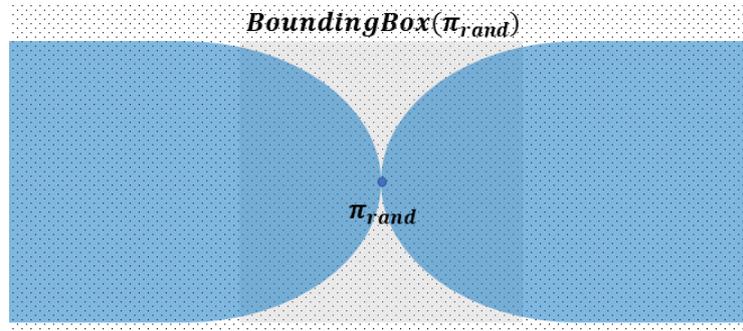


Figure 5.4: Near Nodes Selection

In the case in which a trajectory starting from  $s_{rand}$  must be defined, since the cost function is not symmetric, a second procedure must be introduced. Algorithm 6 shows the main steps of the function **NearNodesToGo**.

As already discussed for the previous problem, the set of reachable states are defined as:

$$N_{ReachToGo} = \{s \in N : \text{dist}(s_{rand}, s) \leq \bar{d}(n)\}$$

Similarly, the set of the neighbour nodes, for the trajectories starting from  $s_{rand}$ , is:

$$N_{NearToGo} = N_{ReachToGo} \cap \text{BoundingBox}(\pi_{rand})$$

**Algorithm 6** NearNodesToGo( $s_{rand}$ )

---

```

1:  $N_{near} \leftarrow 0$ ;
2:  $\pi_{rand} \leftarrow \mathbf{GetPosition}(s_{rand})$ 
3: for  $\forall s \in N$  do
4:   if  $s \in \mathbf{BoundingBox}(\pi_{rand})$  then
5:      $\sigma_{s_{rand} \rightarrow s} \leftarrow \mathbf{FindTrajectory}(s_{rand}, s)$ 
6:     if  $Cost(\sigma_{s_{rand} \rightarrow s}) \leq \bar{d}(n)$  then
7:        $N_{near} \leftarrow N_{near} \cup \{s\}$ 
8:     end if
9:   end if
10: end for
11: return  $N_{near}$ 

```

---

**Best Parent**

Algorithm 7 shows the main steps of the function **FindBestParent**.

**Algorithm 7** FindBestParent( $s_{rand}, N_{near}$ )

---

```

1:  $c_{best} \leftarrow \mathbf{inf}$ ;
2: for  $\forall s_{near} \in N_{near}$  do
3:    $\sigma_{near}, c_{near} \leftarrow \mathbf{FindTrajectory}(s_{near}, s_{rand})$ 
4:   if  $\mathbf{isCollisionFree}(\sigma_{near})$  then
5:     if  $Cost(\rightarrow s_{near}) + c_{near} < c_{best}$  then
6:        $s_{best} \leftarrow s_{near}; e_{best} \leftarrow \{s_{near}, s_{rand}\}$ 
7:        $c_{best} \leftarrow Cost(\rightarrow s_{near}) + c_{near}$ 
8:     end if
9:   end if
10: end for
11: return  $s_{best}, e_{best}$ 

```

---

Aim of this procedure is to find the node which can be recognized as the best parent to be connected with  $s_{rand}$ . In particular, once a set of near nodes  $N_{near}$  has been retrieved from the previously described procedure, a set of trajectories is generated in order to find the best collision-free path to be used as an edge. In particular, the parent node  $s_{best}$  is selected as:

$$s_{best} = \arg \min (Cost(\rightarrow s_{near}) + Cost(\sigma_{s_{near}, s_{rand}}))$$

provided that the trajectory  $\sigma_{s_{near} \rightarrow s_{rand}}$  is collision-free.

It is important to remember that each trajectory is not computed on line, but it is extracted from the database of motion primitives, through the function **FindTrajectory**. Figure 5.5 shows the main steps involved in the procedure used to explore the database:

- The initial state  $s_0$  and the final one  $s_f$  are first normalized in order to make the position vector  $\pi_0$  of the state  $s_0$  coincident with the origin, by applying the translation invariance property. The new pair is then expressed as  $(\tilde{s}_0, \tilde{s}_f)$ .
- A research inside the database is done to retrieve the trajectory connecting the two points of the the grid coincident with the pair  $(\tilde{s}_0, \tilde{s}_f)$ . An index is used in order to identify the extracted elements. Therefore, the trajectory  $\sigma_{index}$  and the cost  $c_{index} = Cost(\sigma_{index})$  are retrieved.
- The inverse of the geometric transformation applied in the first step is used to recover the trajectory  $\sigma_{s_0 \rightarrow s_f}$ , connecting the original pair of states  $(s_0, s_f)$ .

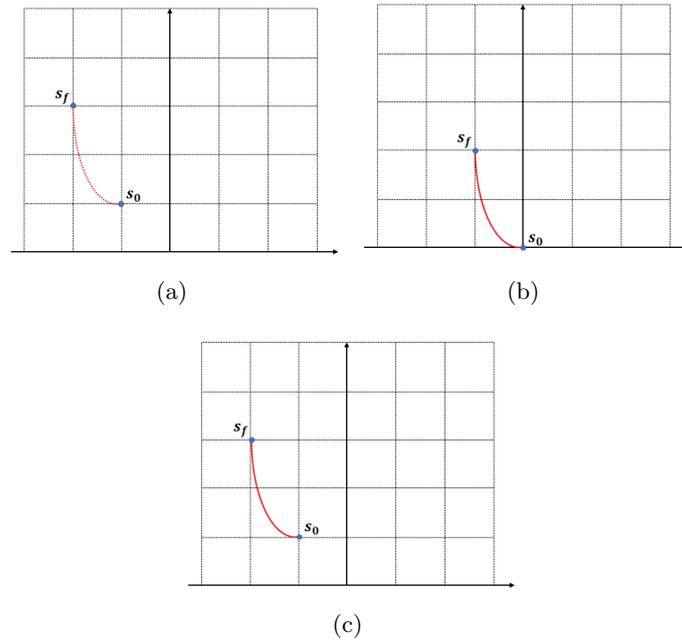


Figure 5.5: (a) Original Pair (b) Translation to the Origin (c) Inverse Transformation

### Update Connection

As already discussed, one of the main differences between RRT\* Motion Primitives and the other sample-based approaches is related to the fact that, instead of extracting a discrete sample from a continuous State Space, it will select a random discrete state from a precomputed grid. This procedure brings to a non-zero probability to sample again a state already belonging to the tree.

Algorithm 8 describes the **UpdateNew** procedure, which deals with the situation in which a random sample is selected for the first time. In this particular case, the procedure simply updates the tree by adding the state  $s_{rand}$  and the optimal edge retrieved from the **FindBestParent** procedure.

---

**Algorithm 8** UpdateNew( $s_{rand}, s_{best}, e_{best}$ )

---

- 1:  $N \leftarrow N \cup s_{rand}$
  - 2:  $E \leftarrow E \cup \{s_{best}, s_{rand}\}$
  - 3: **return**  $N, E$
- 

If instead a selected random sample already belongs to the tree, the **UpdateExisting** procedure, described by Algorithm 9, will be considered. In this case, the function retrieves the parent node  $s_{parent}$  of  $s_{rand}$  and checks if it coincides with  $s_{best}$ . If the two nodes are different, this means that the **FindBestParent** procedure has selected a node as a candidate parent for  $s_{rand}$  which is better than the previous one. For this reason, it is necessary to remove the previous edge, connecting  $s_{parent}$  with  $s_{rand}$ , in order to add the best one  $e_{best}$ .

---

**Algorithm 9** UpdateExisting( $s_{rand}, s_{best}, e_{best}$ )

---

- 1:  $s_{parent} \leftarrow \mathbf{Parent}(s_{rand})$
  - 2: **if**  $s_{parent} \neq s_{best}$  **then**
  - 3:    $e_{prev} \leftarrow \{s_{parent}, s_{rand}\}$
  - 4:    $E \leftarrow \{E \setminus e_{prev}\} \cup \{e_{best}\}$
  - 5: **end if**
  - 6: **return**  $N, E$
- 

## Rewire

The **Rewire** function is the same used by RRT\*. It is important to recall that, in order to retrieve the optimal solution, this procedure is fundamental. In particular, for a given iteration  $i$ :

$$c^* \leq Cost(\rightarrow s_{goal,i})^{RW} \leq Cost(\rightarrow s_{goal,i})^{noRW}$$

where  $c^*$  is the optimal solution for a given path planning problem, while  $Cost(\rightarrow s_{goal,i})^{RW}$  and  $Cost(\rightarrow s_{goal,i})^{noRW}$  are the cumulative costs to reach the goal region, for a given iteration  $i$ , respectively by applying the **Rewire** procedure or without it. Algorithm 10 shows the main steps of the function **Rewire**.

The set of nearby nodes  $N_{near}$  with respect to  $s_{rand}$  is searched with the function **NearNodesToGo**. For each  $s_{near} \in N_{near}$ , it is checked if the trajectory connecting  $s_{rand}$  and  $s_{near}$  is collision-free and, moreover, if:

$$Cost(\rightarrow s_{rand}) + Cost(\sigma_{s_{rand} \rightarrow s_{near}}) \leq Cost(\rightarrow s_{near})$$

**Algorithm 10** Rewire( $s_{rand}, N_{near}$ )

---

```

1: for  $\forall s_{near} \in N_{near}$  do
2:    $\sigma_{near}, c_{near} \leftarrow \mathbf{FindTrajectory}(s_{rand}, s_{near})$ 
3:   if  $Cost(\rightarrow s_{rand}) + c_{near} < Cost(\rightarrow s_{near})$  then
4:     if  $\mathbf{isCollisionFree}(\sigma_{near})$  then
5:        $s_{parent} \leftarrow \mathbf{Parent}(s_{near})$ 
6:        $e_{prev} \leftarrow \{s_{parent}, s_{rand}\}$ 
7:        $e \leftarrow \{s_{rand}, s_{near}\}$ 
8:        $E \leftarrow \{E \setminus e_{prev}\} \cup \{e\}$ 
9:     end if
10:  end if
11: end for
12: return  $N, E$ 

```

---

In the case in which the above condition is true, the tree will be “rewired”, through the procedure:

$$E = \{E \setminus e_{prev}\} \cup \{e\}$$

where  $e = \{s_{rand}, s_{near}\}$ , while  $e_{prev}$  is the previous computed edge connecting  $s_{rand}$  and  $s_{near}$ .

**Best Path**

As already discussed, the property of probabilistic completeness ensures to find a solution only asymptotically. This means that, if a feasible solution cannot be found, the algorithm will run continuously without stopping. For this reason, it is important to set a maximum number of iterations  $n_{iter}$ , in order to force the algorithm to stop at a certain instant. At the end of iterations, i.e. when  $i = n_{iter}$ , the tree will be completed and, through the **BestTraj** procedure, the best trajectory will be extracted. It is important to note that, since the RRT\* Motion Primitives is only asymptotically optimal, this means that the selected maximum number of iterations may not ensure that the best trajectory coincides with the optimal one. For this reason, the number of iterations must be chosen with a high value. With the same idea, if an optimal solution has been found before the end of the algorithm, the subsequent iterations will not bring any improvement.

**5.5 RRT\* MP on Probabilistic Maps**

The main contribution given in this thesis is related to the fact that the RRT\* Motion Primitives can be easily used to handle a crowded environment described through the use of a probabilistic map. In particular, in this work

it is assumed that a Binary Grid is used by the planner to check if a given trajectory is collision-free.

Occupancy Grids are a particular representation of a robot workspace, which are very common in robotics algorithms such as path planning. Planar Grids are retrieved by applying a discretization on a simple 2D map of a given environment. In particular, each element of a Binary Occupancy Grid can assume only two values, in order to express if a given discrete element is occupied or not. Binary Occupancy Grids can be used in the approaches related to the collision checking, during path planning.

To this aim, the function **isCollisionFree**, already introduced, receives in input a given trajectory, which will be discretized into many small intervals. For each point defining each section, it will be checked if at least one lies inside an obstacle, as shown in Figure 5.6. If the previous condition is not satisfied, then the considered trajectory is not collision-free.

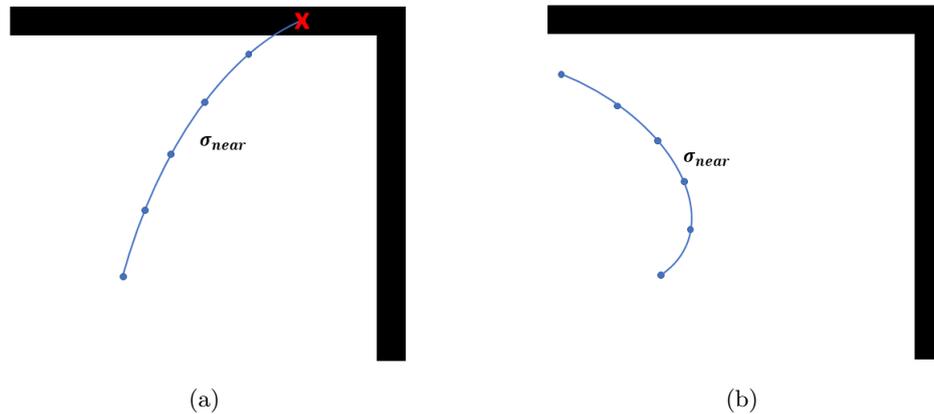


Figure 5.6: Collision Check (a) No Collision-Free Path (b) Collision-Free Path

However, a second class of discrete grids, which goes under the name of Probability Occupancy Grids, can be introduced. This particular representation allows to characterize a robot workspace in a more detailed way. Each cell of the Occupancy Grid is characterized by a value which expresses the probability of its occupancy. In particular, values close to 0 express a high probability to find a free path, while values close to 1 represent a high probability for the cell to contain an obstacle. Figure 5.7 shows the main difference between Binary and Probabilistic Occupancy Grids.

Different grey levels in the probabilistic map express different values of probability. From now on, the discussion will be developed under the hypothesis that the characterization of each crowd, already described in Chapter 3, is available for the planner. To this aim, a darker grey area inside the map expresses a higher concentration of humans, with respect to a lighter grey area, which can be interpreted as a less populated one. The main ad-

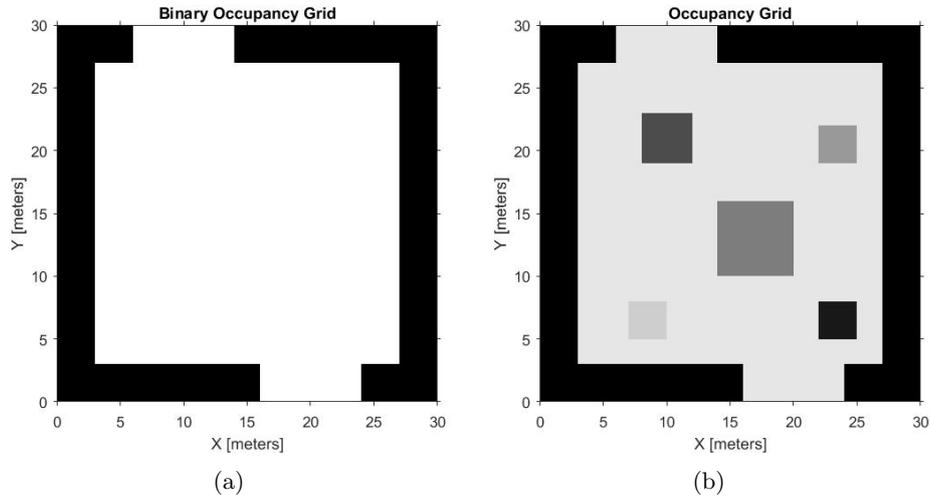


Figure 5.7: Occupancy Grids in MATLAB: (a) Binary: white pixels represent value 0, black pixels represent value 1 (b) Probabilistic: Shades from White to Black represent probabilities from 0.001 to 0.999

vantage in using the RRT\* Motion Primitives approach is that it is possible to simply perform a human-aware path planning, without introducing too much modifications inside the algorithm. This result can be obtained by simply adding a further cost, which will be called *Probabilistic Cost*.

By taking into account a trajectory  $\sigma_{s_1 \rightarrow s_2}$ , the Probabilistic Cost can be defined as the sum of the corresponding probability for each portion of the discretized trajectory  $\sigma_{s_1 \rightarrow s_2}^\Delta$ . Figure 5.8 shows the main steps in order to retrieve a Probabilistic Cost.

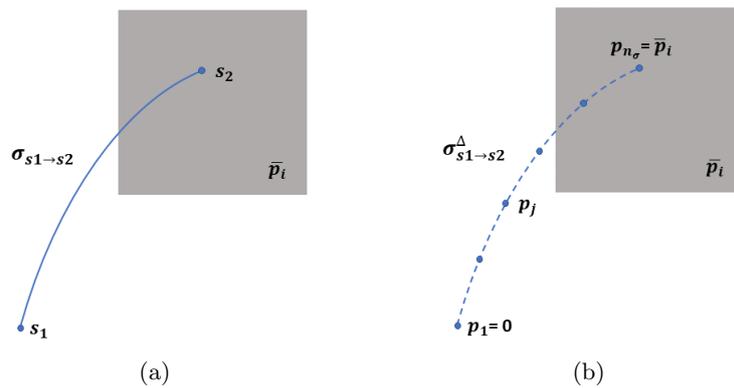


Figure 5.8: (a) Continuous Trajectory (b) Discretized Trajectory

Given a trajectory  $\sigma_{s_1 \rightarrow s_2}$  inside a probabilistic map, it is possible to retrieve a discretized trajectory  $\sigma_{s_1 \rightarrow s_2}^\Delta = \{\sigma_{s_1 \rightarrow s_2}^0, \dots, \sigma_{s_1 \rightarrow s_2}^j, \dots, \sigma_{s_1 \rightarrow s_2}^{n_\sigma}\}$

where  $\sigma_{s_1 \rightarrow s_2}^j$  is the  $j$ -th sample of the discretized trajectory, while  $n_\sigma$  is the maximum number of samples. In order to compute  $\sigma_{s_1 \rightarrow s_2}^\Delta$ , only the information about the  $x$  and  $y$  positions of the trajectory are needed. To this aim, the vectors

$$\xi_{s_1} = (x_1, y_1) \quad \xi_{s_2} = (x_2, y_2)$$

represent the positions of the states  $s_1$  and  $s_2$ . In addition, it is possible to define:

$$\bar{\xi} = \begin{bmatrix} \bar{x} \\ \bar{y} \end{bmatrix}$$

with

$$\bar{x} = \frac{x_2 - x_1}{n_\sigma} \quad \bar{y} = \frac{y_2 - y_1}{n_\sigma}$$

Under the hypothesis that a suitable function  $f_\sigma$  can be used to map each position to corresponding state belonging to  $\sigma_{s_1 \rightarrow s_2}$ , such that:

$$\sigma_{s_1 \rightarrow s_2}^k = f_\sigma(\xi_k)$$

then

$$\sigma_{s_1 \rightarrow s_2}^\Delta = \{f_\sigma(\xi_1), f_\sigma(\xi_1 + \bar{\xi}), \dots, f_\sigma(\xi_1 + j * \bar{\xi}), \dots, f_\sigma(\xi_1 + n_\sigma * \bar{\xi})\}$$

therefore

$$\sigma_{s_1 \rightarrow s_2}^j = f_\sigma(\xi_1 + j * \bar{\xi}) \quad j = 1, \dots, n_\sigma$$

For each sample  $\sigma_{s_1 \rightarrow s_2}^j$  it is possible to define the corresponding probability  $p_j$ . This value coincides with the probability of a grey area, which is supposed to be uniform for each single crowd. With this idea, by defining with  $\bar{p}_i$  the uniform probability of the  $i$ -th area, all the samples  $\sigma_{s_1 \rightarrow s_2}^j$  inside it will have probability  $p_j = \bar{p}_i$ . On the other hand, if a sample  $\sigma_{s_1 \rightarrow s_2}^j$  will be found outside a grey area and inside  $S_{free}$ , then  $p_j$  will be equal to zero. The probability cost is then defined as:

$$c_{prob}(\sigma_{s_1 \rightarrow s_2}) = \sum_{j=1}^{n_\sigma} p_j$$

The presence of a crowd can be simply seen as an additional cost for a feasible trajectory. The function **FindBestParentProb**, highlighted in Algorithm 11, retrieves a set of nearby nodes and, for each element  $s_{near}$ , it extracts a feasible trajectory  $\sigma_{near}$  from the database of motion primitives. In order to retrieve the best parent node, the procedure checks if the given trajectory is collision-free and also if it satisfies the cost requirements.

In addition, the overall cost is computed by taking into account also the probabilistic cost weighted through a constant  $w_p$ , in order to make it comparable to the other costs. It is important to note that this variant takes

**Algorithm 11** *FindBestParentProb*( $s_{rand}, N_{near}$ )

---

```

1:  $c_{best} \leftarrow \text{inf}$ 
2: for  $\forall s_{near} \in N_{near}$  do
3:    $\sigma_{near}, c_{near} \leftarrow \text{FindTrajectory}(s_{near}, s_{rand})$ 
4:    $c_{prob} \leftarrow \text{ProbabilisticCost}(\sigma_{near})$ 
5:   if  $\text{Cost}(\rightarrow s_{near}) + c_{near} + w_p c_{prob} < c_{best}$  then
6:     if  $\text{isCollisionFree}(\sigma_{near})$  then
7:        $s_{best} \leftarrow s_{near}; c_{best} \leftarrow \{s_{near}, s_{rand}\}$ 
8:        $c_{best} \leftarrow \text{Cost}(\rightarrow s_{near}) + c_{near} + w_p c_{prob}$ 
9:     end if
10:  end if
11: end for

```

---

into account the trajectory already extracted from the database, in order to compute the probabilistic cost. With this strategy, the performance of the search-based algorithm will not be affected.

The main drawback with this procedure is related to the fact that it is not possible to consider only a Probabilistic Occupancy Grid. In fact, for sake of simplicity, the function **isCollisionFree** has been designed in order to accept only binary values. For this reason, the global planner will take into account a layered map, in which one layer will contain a Binary Occupancy Grid in order to check if a given trajectory is collision-free, while the second one will be described by a Probabilistic Occupancy Grid, which will be used to compute the Probability Cost, as highlighted in Figure 5.9.

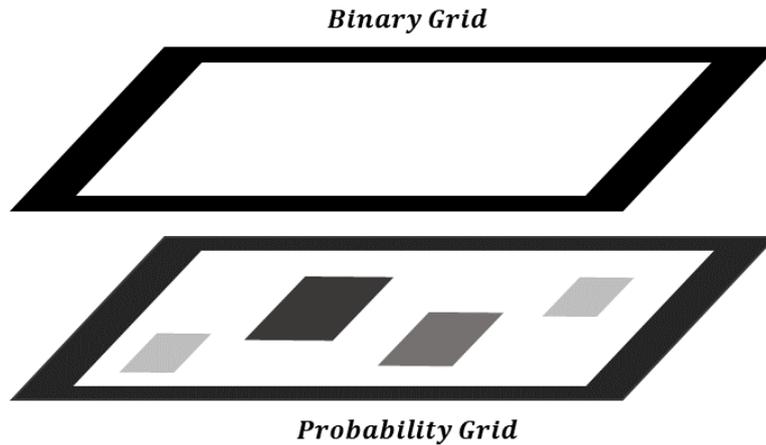


Figure 5.9: Layered Map



## Chapter 6

# Simulations Results

In this chapter, the numerical results, obtained from simulations are reported to show the effectiveness of the proposed algorithms. All the simulations are performed on an IntelCore i7 @2.9GHz personal computer with 12GB RAM and are implemented in MATLAB/SIMULINK. In particular, two main simulations have been realized:

- a MATLAB script to simulate the motion of a particle, representing the model of the vehicle after the introduction of the Feedback Linearization approach. The model is then represented as:

$$\begin{cases} x_P(k+1) = x_P(k) + \tau_{mpc} v_{Px}(k) \\ y_P(k+1) = y_P(k) + \tau_{mpc} v_{Py}(k) \end{cases}$$

from which:

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} \tau_{mpc} & 0 \\ 0 & \tau_{mpc} \end{bmatrix}$$

- a MATLAB script to simulate the approach adopted for global planning, whose result will be an optimal collision-free trajectory, starting from an initial state and ending inside a goal region.

### 6.1 Local Planner

The simulated local planner is summarized in pseudo-code in Algorithm 12 and 13.

For sake of simplicity, the functions **Compute** summarize the overall procedure, described in Chapter 4, to compute all the parameters involved in the MPC formulation. The inputs of the simulated local planner are the initial position of the robot, the goal to be reached and vectors of data structures called *pedestrians* and *obstacles*. Each structure of *pedestrians*

**Algorithm 12** Local Planning

---

```

1: Input : pedestrians, obstacles, position1, goal
2: for  $k = 1 : N$  do
3:    $[u_k, U_k] \leftarrow \text{MPC}(goal, position_k, obstacles, pedestrian, U_{k-1})$ 
4:    $v.x_k \leftarrow u_k(1)$ 
5:    $v.y_k \leftarrow u_k(2)$ 
6:    $position.x_k \leftarrow position.x_{k-1} + v.x_k * \tau_{mpc}$ 
7:    $position.y_k \leftarrow position.y_{k-1} + v.y_k * \tau_{mpc}$ 
8: end for

```

---

**Algorithm 13** MPC(*goal, position<sub>k</sub>, obstacles, pedestrians, U<sub>k-1</sub>*)

---

```

1:  $H \leftarrow \text{Compute\_Hessian}(goal)$ 
2:  $f \leftarrow \text{Compute\_Gradient}(goal)$ 
3:  $[A_{ped}, b_{ped}] \leftarrow \text{Compute\_Pedestrian\_Constraints}(pedestrians, U_{k-1})$ 
4:  $[A_{obs}, b_{obs}] \leftarrow \text{Compute\_Obstacle\_Constraints}(obstacles, U_{k-1})$ 
5:  $A_{pos} \leftarrow [A_{ped}; A_{obs}]$ 
6:  $b_{pos} \leftarrow [b_{ped}; b_{obs}]$ 
7:  $[A_{vel}, b_{vel}] \leftarrow \text{Compute\_Velocity\_Constraints}(position_k)$ 
8:  $[A_{varvel}, b_{varvel}] \leftarrow \text{Compute\_Vel\_Variation\_Constraints}(position_k)$ 
9:  $A \leftarrow [A_{pos}; A_{vel}; A_{varvel}]$ 
10:  $b \leftarrow [b_{pos}; b_{vel}; b_{varvel}]$ 
11:  $U \leftarrow \text{quadprog}(H, f, A, b)$ 
12:  $u_{RH} \leftarrow [U(1); U(2)]$ 
13: return  $u_{RH}, U$ 

```

---

takes the name of *pedestrian* and contains all the information, about moving obstacles, needed to compute the position constraints. In particular:

$$pedestrians = [pedestrian_1, \dots, pedestrian_n]$$

with

$$pedestrian_i = \begin{cases} x_i & \% \text{ initial } x \text{ position} \\ y_i & \% \text{ initial } y \text{ position} \\ vx_i & \% \text{ velocity toward } x \text{ (assumed always constant)} \\ vy_i & \% \text{ velocity toward } y \text{ (assumed always constant)} \\ sides_i & \% \text{ number of sides of virtual box} \end{cases}$$

With the same idea, the vector *obstacles* represents all the fixed obstacles inside the environment. The data structure *obstacle* is defined as:

$$obstacle_i = \begin{cases} x_i & \% \text{ initial } x \text{ position} \\ y_i & \% \text{ initial } y \text{ position} \\ sides_i & \% \text{ number of sides of virtual box} \end{cases}$$

The function **quadprog** belongs to a MATLAB toolbox focused on optimization. In particular, the function used in this algorithm allows to compute the solution of quadratic programming problems, by simply defining the input matrices expressing the cost function and the constraints in the form:

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

where

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

and

$$A \mathcal{U}(k) \leq b$$

The output of **quadprog** is the vector of control variables inside the entire prediction horizon  $\mathcal{U}(k) = [u(k), \dots, u(k+N-1)]^T$ . For this reason, in order to apply the Receding Horizon strategy, the first two elements of  $U$ , representing the velocities along the x and y directions at the first time instant, are extracted and used to define  $u_{RH}$ . In the following, the numerical results and the plots representing the trajectories computed by the simulated local planner will be presented.

### 6.1.1 Convergence to the Goal

The first step, in order to show the effectiveness of the algorithm, focuses on the ability of the local planner to converge to the goal with presence of fixed obstacles (walls), without pedestrians. To this aim, the following values for the MPC parameters have been selected:

$$q = 1 \rightarrow Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$r = 1 \rightarrow R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$k = -\frac{1}{2\tau_{mpc}} \rightarrow s = \frac{(q + k^2 r)}{1 - (1 + \tau_{mpc} k)^2} = 9.67 \rightarrow S = \begin{bmatrix} 9.67 & 0 \\ 0 & 9.67 \end{bmatrix}$$

As already explained in Chapter 4,  $r$  and  $q$  represent the penalty given to the control variables and to the error on the state, with respect to the reference, respectively. For this reason, the values defined above are selected in order to obtain a good trade-off between the two. The value of  $s$  is then retrieved with the introduced formula to guarantee the asymptotic stability of the system.

In order to choose the values of the prediction horizon  $N$  and the controller sampling time  $\tau_{mpc}$  it is important to recall that:

$$N \tau_{mpc} = T_{ph}$$

where  $T_{ph}$  is the prediction horizon, defined as the time window inside which the controller will predict the system evolution. The main hypothesis made in this work is related to the fact that it is possible to simplify the pedestrian motion as a linear one and with constant velocity inside a short prediction horizon. On the other side, a too much short prediction horizon will reduce the predictive ability of the controller. The compromise can be obtained by setting  $T_{ph}$  equal to 4 s. However, it is important to tune  $N$  and  $\tau_{mpc}$  accordingly. In particular, a small value of  $\tau_{mpc}$  implies a faster reactivity of the controller but, at the same time, a high value of  $N$  means a high computational load. For this reason, the following choice has been done:

$$\tau_{mpc} = 0.2 \text{ s} \quad N = 20$$

Figure 6.1 shows the results.

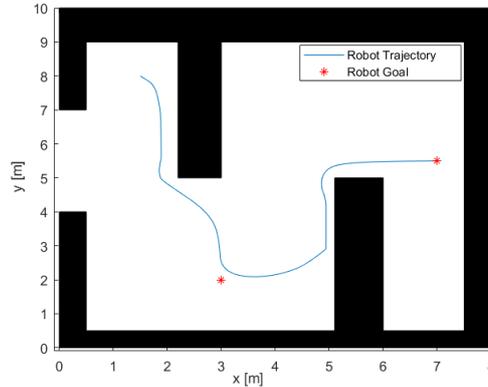


Figure 6.1: Trajectory Toward the Reference

The scenario has been designed in order to represent a room with two walls. The initial position of the vehicle has been set as  $P^0 = [1.5, 8]$ , while two goals have been defined in  $P_{goal1} = [3, 2]$  and  $P_{goal2} = [7, 5.5]$ . The property of convergence is satisfied and, moreover, the trajectory to reach the goal ensures a good obstacles avoidance. It is important to note how, for complex scenarios, more than one goal needs to be defined.

### 6.1.2 Consistency of Velocity and Velocity Variation Constraints

The second step is to ensure that the bounds on velocity and on its variation are satisfied. To this aim, Figure 6.2 shows a second scenario. The parameters related to the constraints have been selected as:

$$v_{max} = 0.7 \text{ [m/s]} \quad a_{max} = 0.35 \text{ [m/s}^2\text{]}$$

The results are shown in Figure 6.3.

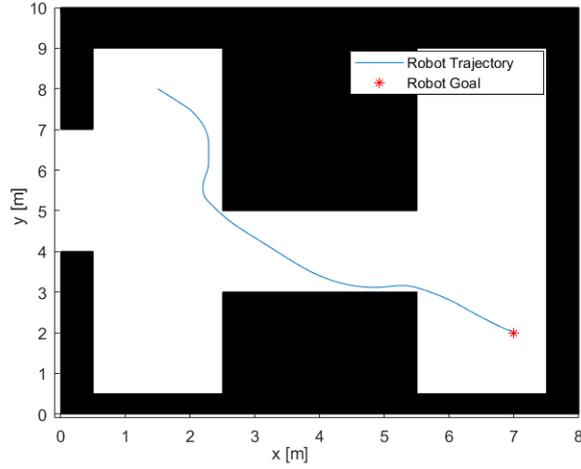


Figure 6.2: Trajectory Toward the Reference

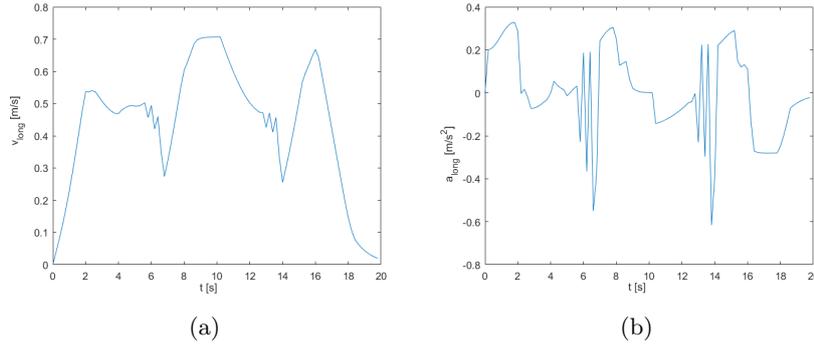


Figure 6.3: (a) Bounded Longitudinal Velocity (b) Bounded Longitudinal Acceleration

### 6.1.3 Pedestrian Avoidance

In order to show the consistency of position constraints related to the pedestrians, it is fundamental to fully characterize them inside the simulation. The first experiment in this context is performed by taking into account only a single pedestrian. The first step is to ensure a correct representation of the pedestrian inside the space, with the virtual box correctly oriented. Figure 6.4 highlights the pedestrian characterization in two different time instants during its motion.

The kinematic of the pedestrian is described by the following discrete time model:

$$\begin{cases} x_{ped}(k+1) = x_{ped}(k) + \tau_{mpc} v_x \\ y_{ped}(k+1) = y_{ped}(k) + \tau_{mpc} v_y \end{cases}$$

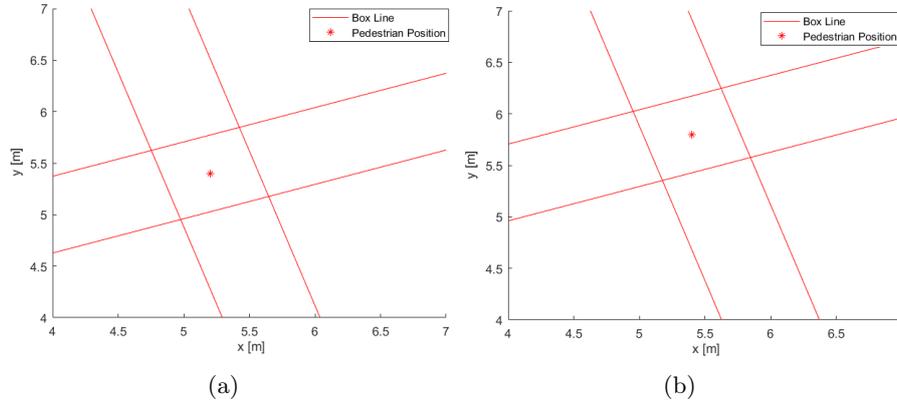


Figure 6.4: Pedestrian with velocity vector  $v_{ped} = [1; 2]$  and Virtual Box with side length of  $0.5m$ .

With the above characterization it is now possible to introduce the results related to the consistency of position constraints, highlighted in Figure 6.5.

The simulation takes into account a pedestrian inside the environment and, in particular, it starts from the initial position  $P_{ped} = [5.5, 5.5]$  and it proceeds toward the robot initial position with a velocity vector  $v_{ped} = [-0.5; -0.5]$ . The modulus of the pedestrian velocity is chosen considering the information retrieved from [20], which identifies as  $2 m/s$  the velocity with which the pedestrian switches from walking to running. For this reason, a velocity equal to  $0.5 m/s$  corresponds to a slow walking, typical of an indoor situation. The vehicle starts from  $P^0 = [2, 2]$  and reaches  $P^{ref} = [6, 6]$ .

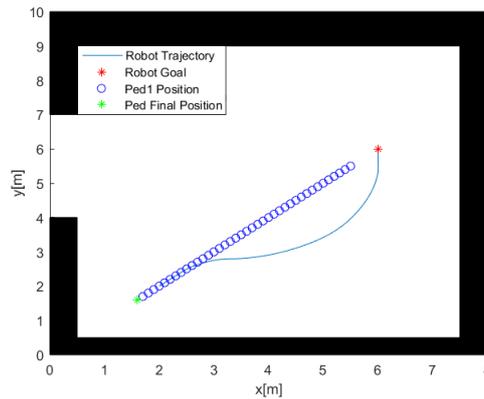


Figure 6.5: Robot and Pedestrian Trajectories

Since the pedestrian walks to the same direction of the optimal trajectory, the robot is forced to deviate in order to avoid collision, by keeping a

safety distance of around  $0.5\text{ m}$ , which coincides with the dimension of the virtual box. In order to check if the avoidance has been guaranteed, Figure 6.6 and 6.7 show the same experiment at different time instants.

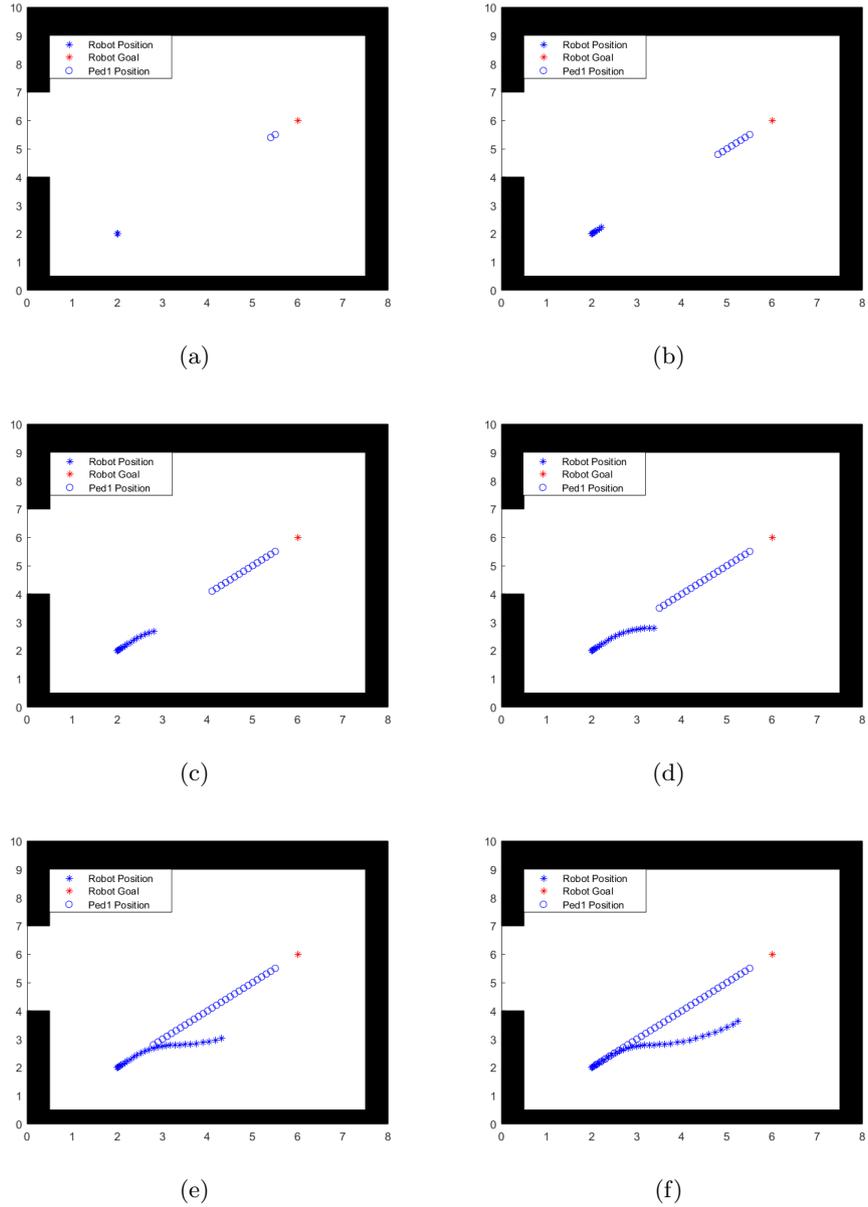


Figure 6.6: Robot and Pedestrian Trajectories

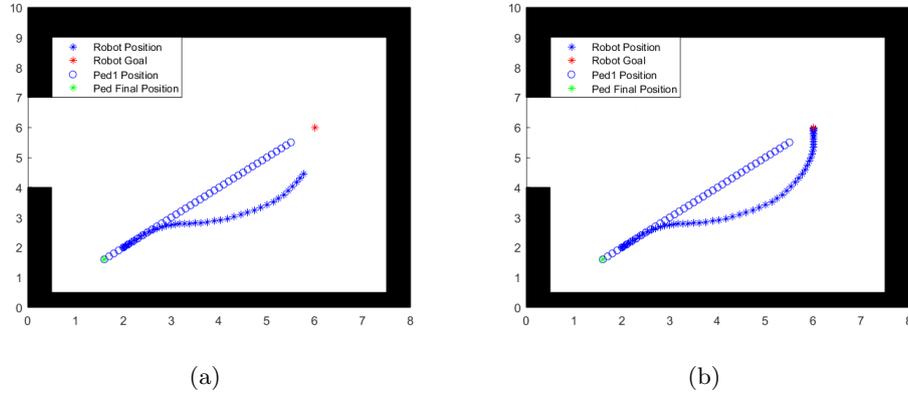


Figure 6.7: Robot and Pedestrian Trajectories

#### 6.1.4 Consistency of Constraints with Slack Variables

The last step is based on checking the effectiveness of the slack variables. By recalling that the variable  $\Delta l$  expresses the reduction of the trust region of the state space, it is chosen in order to take into account the real size of the wheelchair. For this reason, the simulation in Figure 6.8 takes into account  $\Delta l = 0.5 \text{ m}$ .

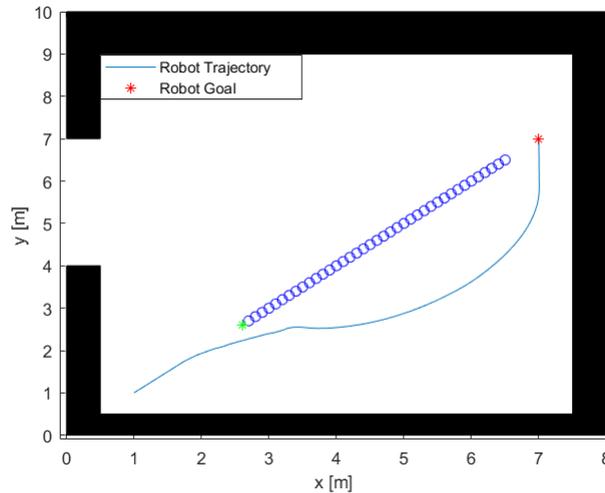


Figure 6.8: Robot and Pedestrian Trajectories with Slack Variables

In contrast, Figure 6.9 shows the same experiment without the introduction of the slack variables.

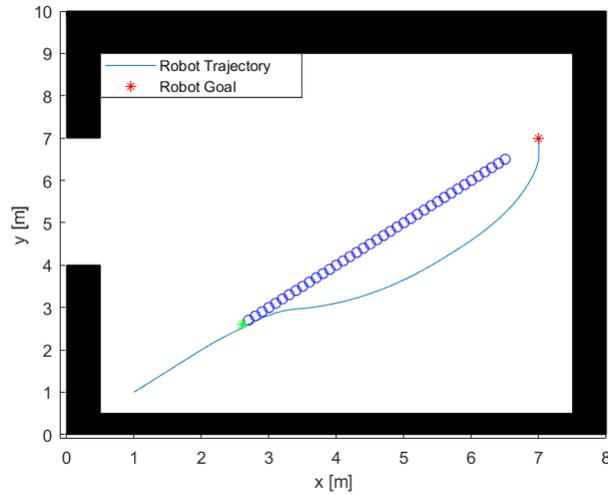


Figure 6.9: Robot and Pedestrian Trajectories without Slack Variables

### 6.1.5 Complete Simulations

In the following, different scenarios will be presented:

- Figure 6.10 shows a simulation performed inside a room with four pedestrians. In particular:
  - Pedestrian1 starts from  $P_{ped1} = [6, 5]$  with velocity vector  $v_{ped1} = [-0.5; 0]$ ;
  - Pedestrian2 starts from  $P_{ped2} = [6.5, 7]$  with velocity vector  $v_{ped2} = [-0.5; -0.3]$ ;
  - Pedestrian3 starts from  $P_{ped3} = [5, 8]$  with velocity vector  $v_{ped3} = [0; -1]$ ;
  - Pedestrian4 starts from  $P_{ped4} = [1.5, 3]$  with velocity vector  $v_{ped1} = [0.9; -0.3]$ ;
  - The vehicle starts from  $P^0 = [2, 4]$  to reach the goal  $P^{ref} = [7; 7]$ ;

Again, in order to check the effective avoidance, different time instants are shown in Figure 6.11. The aim of this simulation is to show that the controller can handle the presence of different pedestrians.

- The second simulation is shown in Figure 6.12. It represents a pedestrian, starting from  $P_{ped} = [5.8, 5]$  with  $v_{ped} = [-0.7; 0]$ , walking toward the vehicle, with  $P^0 = [1.5, 5]$  and  $P^{ref} = [6.5, 5]$ , but this time the motion is performed inside a small space, as a corridor. Aim of this simulation is to show that the local planner can handle at the same time the constraints related to the pedestrians and to the fixed obstacles.

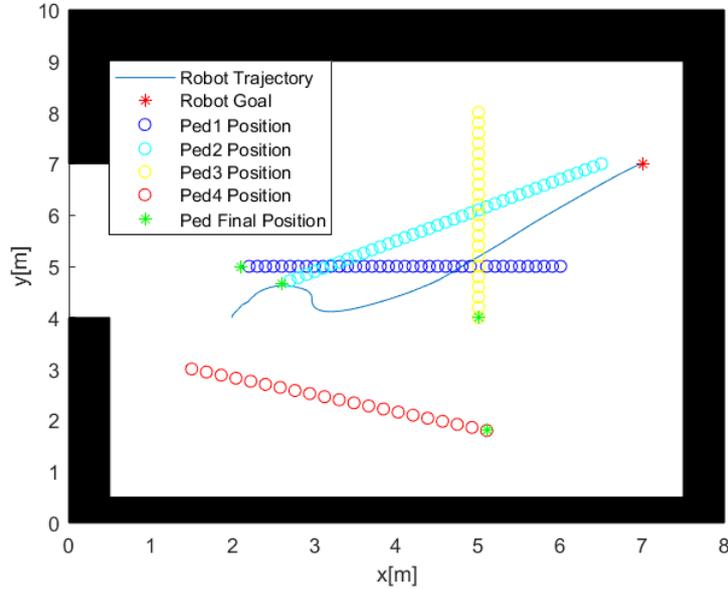


Figure 6.10: Robot and Pedestrians Trajectories

- The last simulation is presented in Figure 6.13. It shows a more complex scenario, in which more than one goal has been defined and the local planner has to take into account both pedestrians and fixed obstacles. This experiment is characterized by the following parameters:
  - Pedestrian1 starts from  $P_{ped1} = [1.5, 7]$  with velocity vector  $v_{ped1} = [0; -0.3]$ ;
  - Pedestrian2 starts from  $P_{ped2} = [4, 7.5]$  with velocity vector  $v_{ped2} = [-0.1; -0.5]$ ;
  - Pedestrian3 starts from  $P_{ped3} = [5.5, 6.5]$  with velocity vector  $v_{ped3} = [0.3; -0.6]$ ;
  - The vehicle starts from  $P^0 = [1.5, 8.5]$  to reach the goals  $P^{ref1} = [1.5; 1.5]$ ,  $P^{ref2} = [4; 1.5]$ ,  $P^{ref3} = [4; 8.5]$ ,  $P^{ref4} = [6.5; 2]$ ;

It is important to note how this time one of the pedestrians does not walk towards the vehicle, but instead it starts in front of it at a very slow velocity, in order to show how the vehicle behaves in that situation. In particular, it follows the same direction of the pedestrian until a safety distance then, since the vehicle is moving faster, it tries to pass in order to reach the goal. Another interesting conclusion can be retrieved from the last part of the vehicle trajectory. In fact, due to the presence of the fixed obstacle and of the third pedestrian, the vehicle decides to choose the longest path in order to avoid collision with both of them.

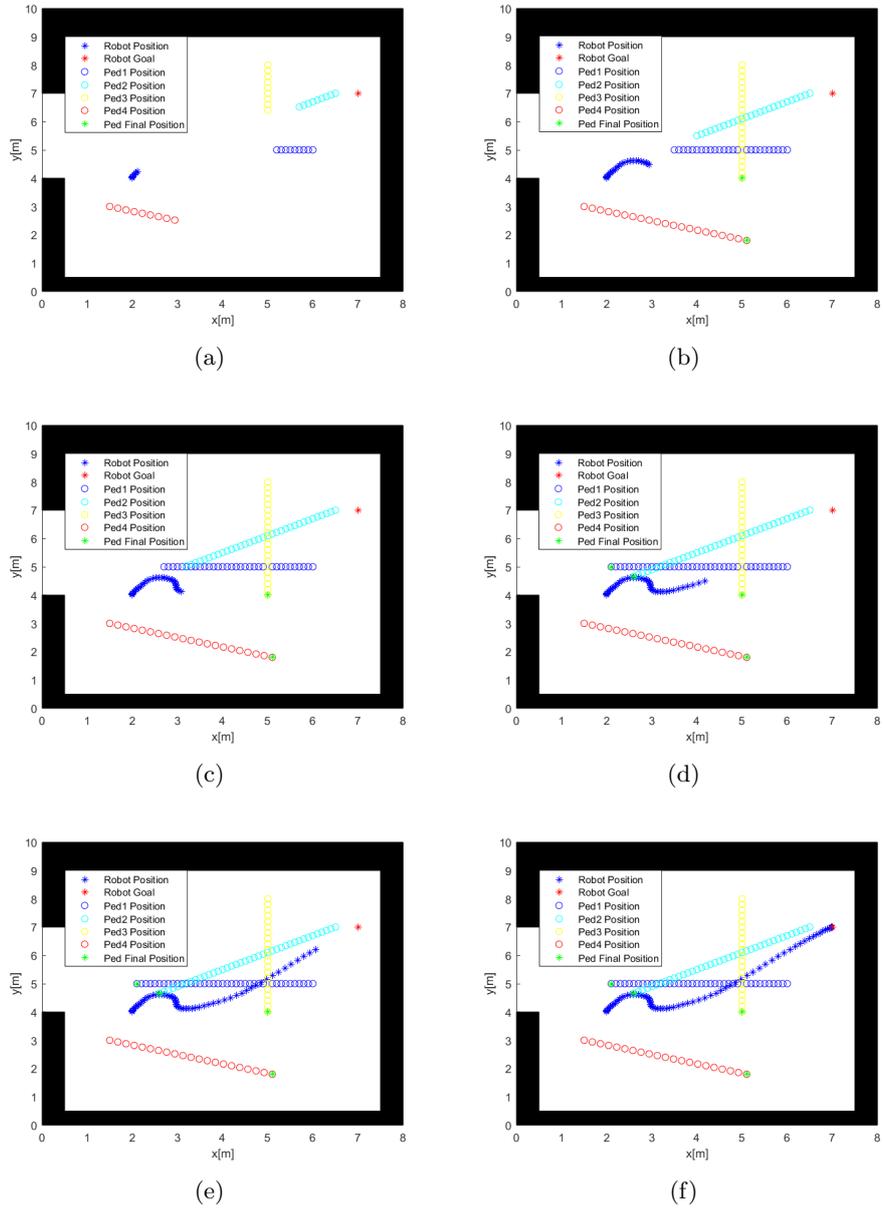


Figure 6.11: Robot and Pedestrian Trajectories

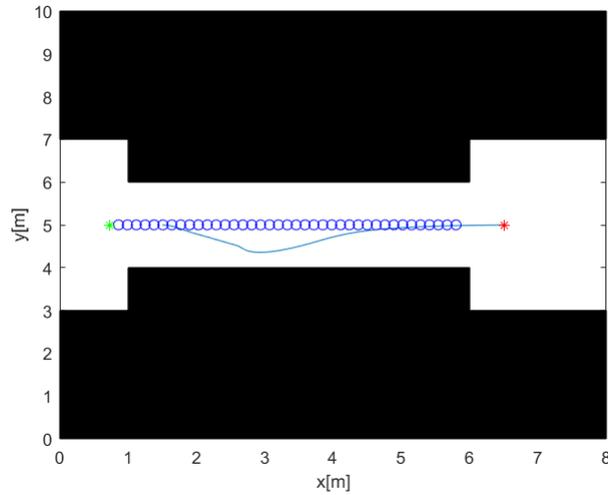


Figure 6.12: Robot and Pedestrian Trajectories

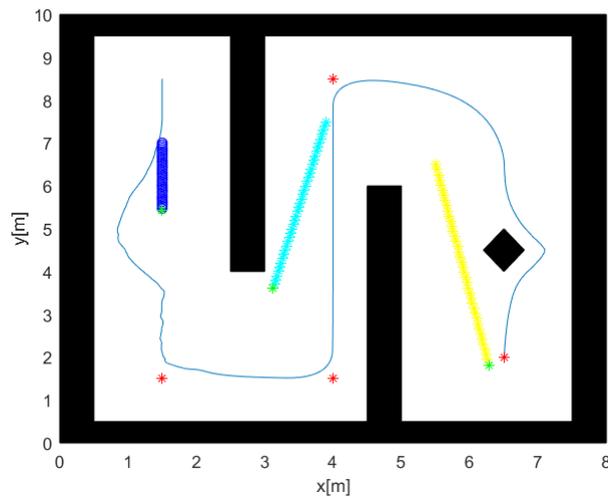


Figure 6.13: Robot and Pedestrians Trajectories

## 6.2 Global Planner

The simulated global planner relies on the definition of two different maps, both representing an example of a building plan. In particular the Binary Occupancy Grid is used in order to check collisions, while the Probabilistic Occupancy Grid allows to compute the probabilistic cost defined in Chapter 5. Figure 6.14 shows the two different maps.

The simulation environment is bounded by  $x \in [0, 20]$  m and  $y \in [0, 20]$  m. Other limitations related to the system are:

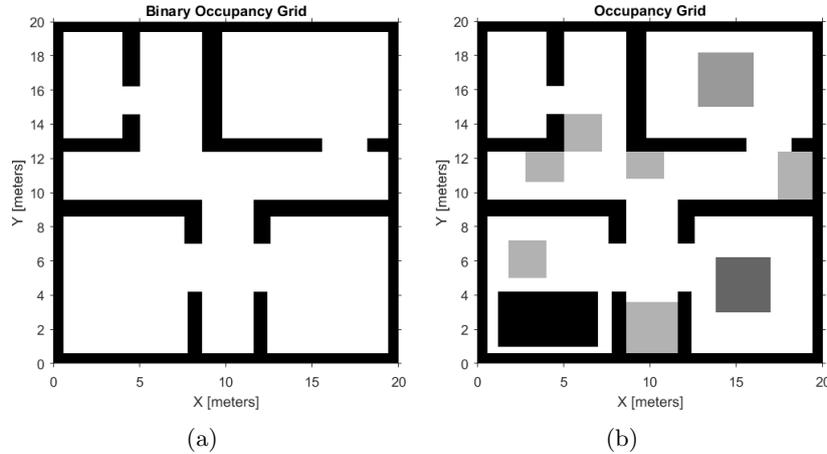


Figure 6.14: Occupancy Grids used for the simulated Global Planner: (a) Binary Occupancy Grid (b) Probabilistic Occupancy Grid

- $\theta \in [0, 2\pi]$  rad
- $v \in [0, 4]$  m/s
- $a \in [-3, 3]$  m/s<sup>2</sup>
- $\omega \in [-3, 3]$  rad/s

The database of motion primitives is taken from the work described in [4]. It is a fine resolution uniform square grid with initial state centred in  $(x_0, y_0) = (0, 0)$  and final state  $(x_f, y_f) \in [-2, 0) \cup (0, 2] \times [-2, 0) \cup (0, 2]$  with a resolution of half a meter (which is the measure of the side of a square cell). The initial and final velocities can have 5 different values in the range  $[0, 4]$  m/s, while the initial and final orientation are selected among 24 different values spaced in the range  $[0, 2\pi)$  rad. Figure 6.15 represents the grid used to define the database.

The first experiment aims at showing the effectiveness of the RRT\* Motion Primitives inside a crowded environment. Figure 6.16 shows the simulation performed with an initial state  $s_0 = [x_0, y_0, \theta_0] = [1, 18, 0]$  and a goal region of  $1 \times 1$  m centred in  $(17.5, 2.5)$  without any goal on orientation. In contrast, Figure 6.17 shows a simulation with the use of a single Binary Occupancy Grid in order to highlight the difference with respect to a crowded context. In both cases, the red line expresses the optimal trajectory computed with the RRT\* Motion Primitives approach. While in the human-free rooms the robot follows the optimal trajectory, which minimizes the execution time, in the crowded context it is forced to deviate in order to avoid the areas with a higher value of probability (the grey squares). In addition, Figure 6.18 shows the time evolution of the longitudinal velocity,

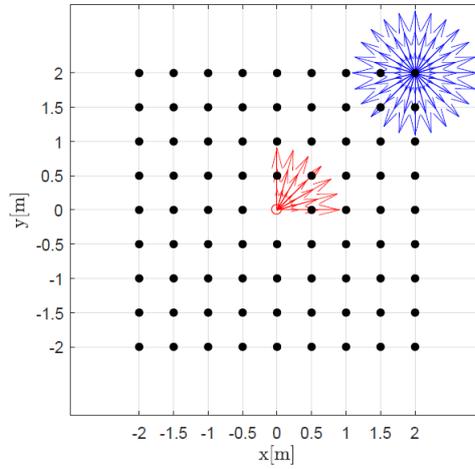


Figure 6.15: "Square Grid with Fine Resolution", pg.55, *Optimal Kinodynamic Planning For Autonomous Vehicles*, B.Sakcak

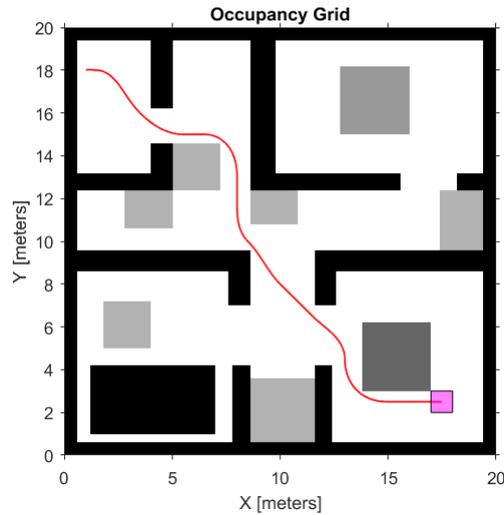


Figure 6.16: *Optimal Trajectory inside a Crowded Environment*

the angular one and the acceleration in order to highlight the satisfaction of the constraints imposed to the problem (red lines inside the plots), by proving the effectiveness of the RRT\* Motion Primitives as a kinodynamic motion planning approach.

The second experiment is based on checking the different results with respect to the number of iterations. In fact, as discussed in Chapter 5, the property of probabilistic completeness ensures to find a solution only asymptotically. This means that, if a feasible solution cannot be found, the algorithm will run continuously without stopping. For this reason, it is important to set a maximum number of iteration  $n_{iter}$ . In the following,

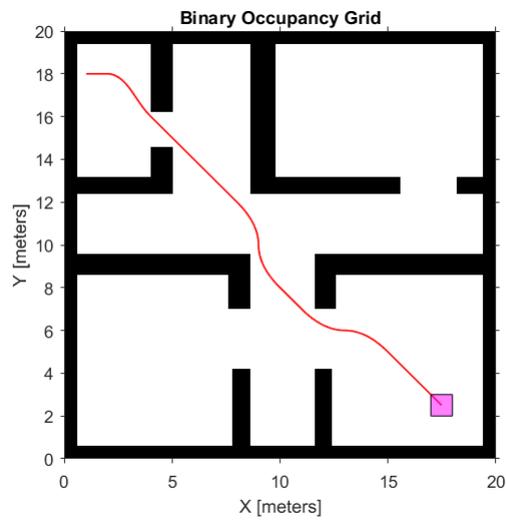


Figure 6.17: Optimal Trajectory inside a Human-free Environment

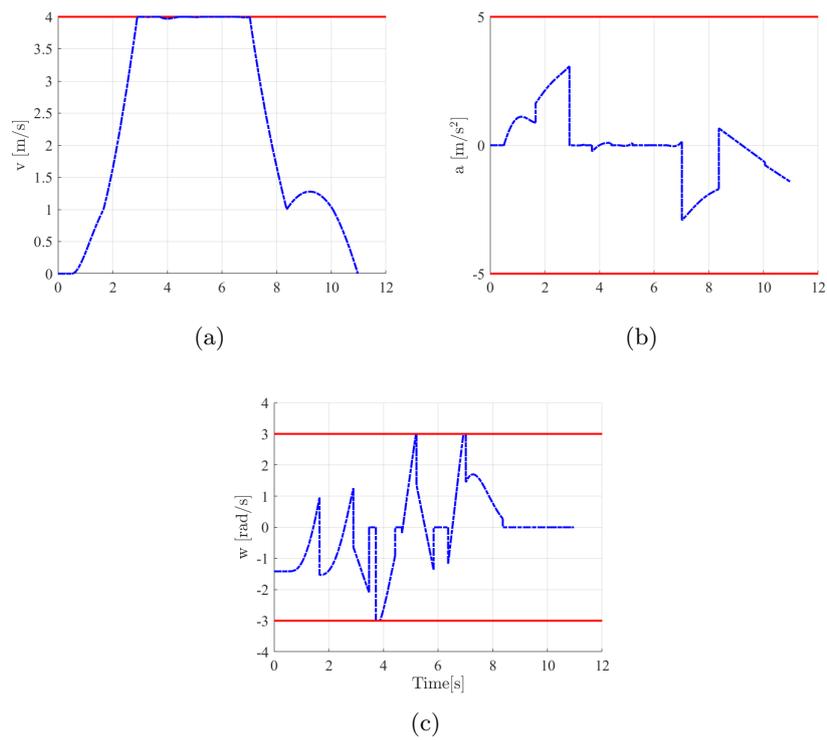


Figure 6.18: Velocity and Actuation Limits: (a) Linear Velocity (b) Acceleration (c) Angular Velocity

different numbers of maximum iteration are used in the same scenario to show the different results. In particular, the maximum number of nodes in the tree have been constrained to the number of iteration through the definition:

$$Nodes_{max} = 2 * Iterations_{max}$$

Figure 6.19 shows the results in the different simulations. As the number of iterations increases, the solution gets closer to the optimal one, until a limit value is reached (in this case it coincides with  $Iterations_{max} = 700000$ ). Up to that value, the solution coinciding with the optimal one will not change.

The last experiment has been designed to show the characterization of the trajectory in the case in which the robot is forced to pass through a probabilistic area. In fact, it may happen that an area occupied by humans may coincide with the only available path for the robot so, in order to reach the goal or to avoid collisions, the trajectory shall traverse it. Figure 6.20 shows three main scenarios.

It is clear how the robot deviates its trajectory with respect to the area in which it is forced to perform its motion. In particular, it is possible to note how the robot modifies the trajectory in order to reach areas with a lower probability cost in a faster way. This situation is clear by looking at the second example. In fact, when the robot approaches to the dark grey zone, it changes orientation in order to proceed in vertical direction. With this strategy, the robot will pass through it in a faster way. This result finds explanation in the fact that, since a darker colour identifies a high probability to encounter a human, in order to reduce this probability, the only way is to follow the shorter path to exit from that area.

In conclusion, the last simulation represents a complete experiment, starting from the computation of the optimal trajectory performed by the global planner until the convergence of the vehicle, controlled by the local planner, to the goal region. Figure 6.21 shows the main steps:

- the optimal trajectory is computed by the global planner;
- the trajectory is then segmented and, for every point of each segment, a goal for the control problem is defined;
- the local planner receives the goals and, by solving a regulation problem, brings the vehicle to each goal, keeping a safety distance from fixed obstacles and pedestrians. The first image related to the control shows the regulation problem without the human presence. It is important to note how the real trajectory of the vehicle is slightly different from the one computed by the global planner. This is due to the fact that the control problem also takes into account the constraints related to the obstacles and, in order to keep a safety distance, it performs a slightly different motion. The second image shows the same

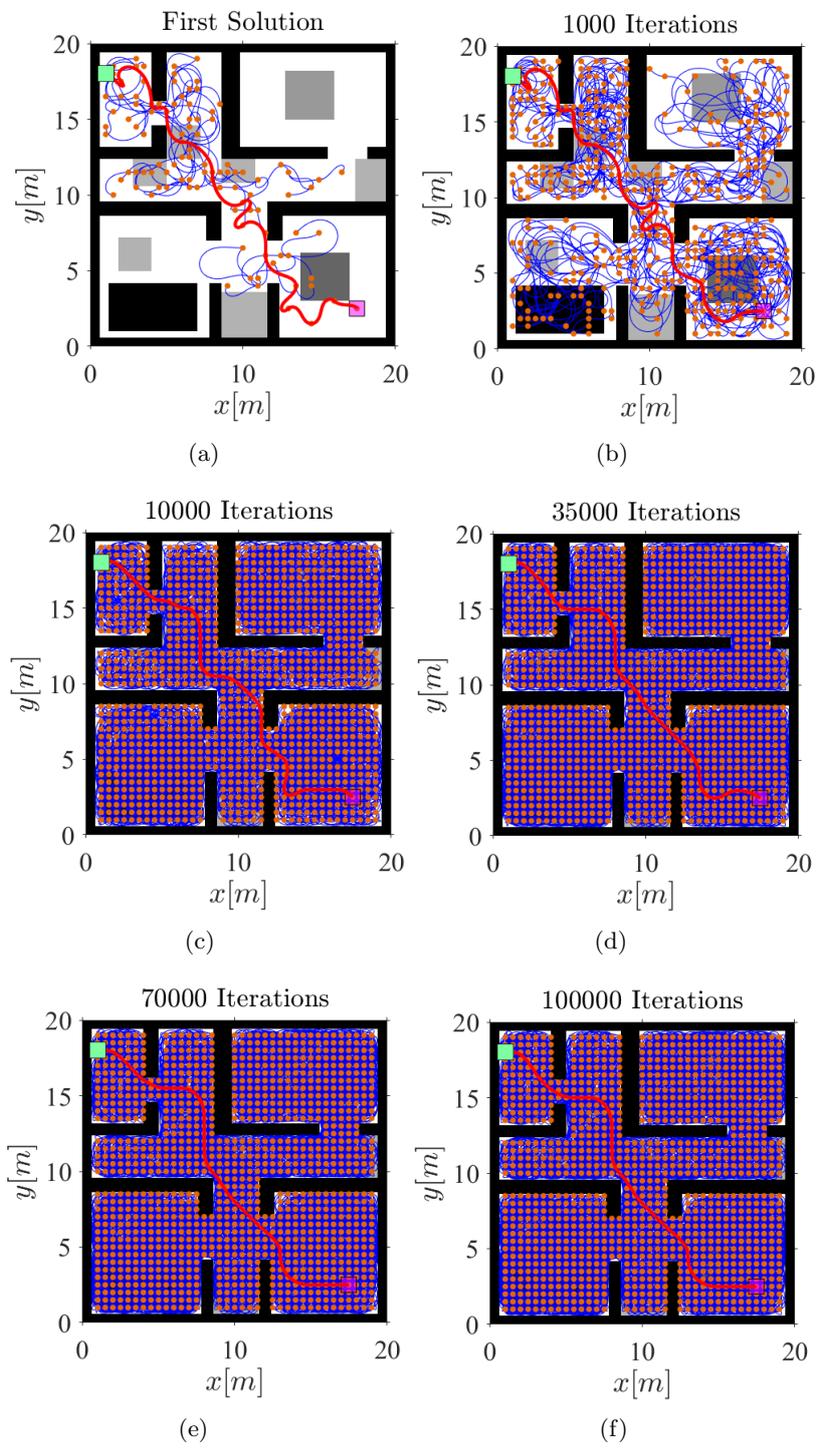


Figure 6.19: Simulations Results for various number of Maximum Iterations



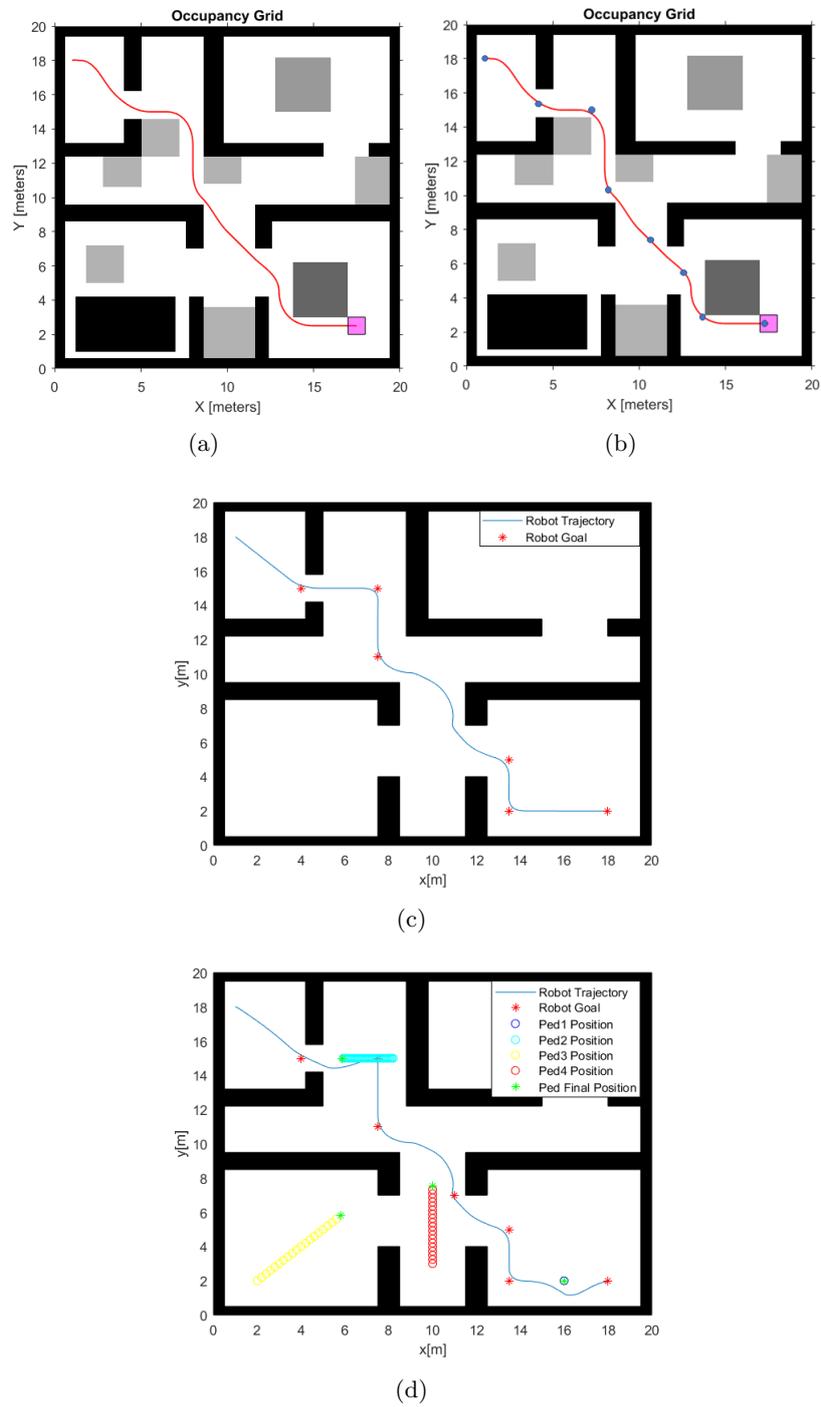


Figure 6.21: Main steps involved in Planning and Control of an Autonomous Wheelchair inside a Crowded Environment



## Chapter 7

# Conclusions

In this thesis a novel approach related to the field of human-aware robot navigation has been presented. The work here described relies on the coordination between a planning approach, like the RRT\* Motion Primitives, and an advanced control strategy, like the MPC.

In particular, the simulations have shown as the motion of a human-aware autonomous vehicle can be efficiently controlled by simply integrating a local planning procedure, which already guarantees obstacle avoidance, with a human-aware trajectory planning. In this way, since the global planner computes a solution that ensures the avoidance of areas with a high concentration of humans, the local planner workload is significantly reduced.

In the context of global planning, the main drawback related to the presented strategy is the performance reduction in terms of execution time. In fact, the introduction of a probabilistic map and the related computations highly reduce the velocity of the algorithm needed to construct the tree. This is due to the fact that, in order to compute the edges characterized by the lowest probabilistic cost, the planning algorithm is forced to explore a larger number of nodes at each iteration, with respect to the canonical case. For this reason, an important upgrade to the global planning strategy can be guaranteed with the reduction of the computation time of the RRT\* Motion Primitives approach on probabilistic maps.



# Bibliography

- [1] A.Turnwald et al. “Interactive Navigation of Humans from a Game Theoretic Perspective”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014)* (2014).
- [2] B.Cohen, S.Chitta, and M.Likhachev. “Search-based planning for manipulation with motion primitives”. In: *IEEE International Conference on Robotics and Automation* (2010).
- [3] B.Donald et al. “Kinodynamic motion planning”. In: *Journal of the ACM (JACM)*, 40(5):1048–1066 (1993).
- [4] B.Sakcak. “Optimal Kinodynamic Planning for Autonomous Vehicles”. PHD thesis. Politecnico di Milano, 2017.
- [5] B.Sakcak et al. “Using motion primitives to enforce vehicle motion constraints in sampling-based optimal planners”. In: *IEEE International Symposium on Circuits and Systems (ISCAS)* (2018).
- [6] B.Siciliano et al. *Robotics: Modelling, Planning and Control, 3rd Ed.* Ed. by Springer. 2009.
- [7] D.J.Webb and J.van den Berg. “Kinodynamic rrt\*: Asymptotically optimal motion planning for robots with linear dynamics”. In: *IEEE, International Conference on Robotics and Automation (ICRA)* (2013).
- [8] E.Ceravolo and M.Gabellone. *Controllo del Moto di una Sedia a Rotelle Autonoma Mediante Tecniche di Controllo Predittivo*. Master’s thesis. 2016.
- [9] E.Hall. *The Hidden Dimension*. Ed. by Anchor Books. 1966.
- [10] E.Rimon and D.E.Koditschek. “The construction of analytic diffeomorphisms for exact robot navigation on star worlds”. In: *IEEE International Conference on Robotics and Automation, Scottsdale, AZ, pp. 21–26, 1989* (1989).
- [11] J.M.Maciejowski and C.Kerrigan. “Soft Constraints and Exact Penalty Functions in Model Predictive Control”. In: *UKACC International Conference (Control 2000)* (2000).

- 
- [12] J.T.Schwartz and M. Sharir. “On the ‘piano movers’ problem: II. General techniques for computing topological properties of real algebraic manifolds”. In: *Advances in Applied Mathematics*, vol. 4, pp. 298–351 (1983).
- [13] L.E.Kavraki et al. “Probabilistic roadmaps for path planning in high-dimensional configuration spaces”. In: *IEEE Transactions on Robotics and Automation*, vol. 12, pp. 566–580 (1996).
- [14] L.Magni and R.Scattolini. *Advanced and Multivariable Control*. Ed. by Pitagora Editrice Bologna. 2014.
- [15] N.A.Nemade and V.V.Gohokar. “A Survey of Video Datasets for Crowd Density Estimation”. In: *International Conference on Global Trends in Signal Processing, Information Computing and Communication* (2016).
- [16] O.Khatib. “Real-time obstacle avoidance for manipulators and mobile robots”. In: *International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98 (1986).
- [17] P.Bolzern, R.Scattolini, and N.Schiavoni. *Fondamenti di Controlli Automatici*. Ed. by McGraw-Hill. 2008.
- [18] P.Fiorini and Z.Shiller. “Motion Planning in Dynamic Environments Using Velocity Obstacles”. In: *The International Journal of Robotics Research Vol.17*, pp. 760-772, Sage Publications Inc. (1998).
- [19] P.Gardziński et al. “Crowd Density Estimation Based on Voxel Model in Multi-View Surveillance Systems”. In: *2015 International Conference on Systems, Signals and Image Processing (IWSSIP)* (2015).
- [20] S.B.Liu et al. “Provably Safe Motion of Mobile Robots in Human Environments”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2017).
- [21] S.Karaman and E.Frazzoli. “Optimal Kinodynamic Motion Planning using Incremental Sampling-based Methods”. In: *IEEE Conference on Decision and Control* (2010).
- [22] S.M.LaValle and J.J. Kuffner. “Rapidly-exploring random trees: Progress and prospects”. In: *New Directions in Algorithmic and Computational Robotics B.R. Donald, K. Lynch, D. Rus, (Eds.), AK Peters, Wellesley, MA, pp. 293–308* (2001).
- [23] S.Misiano. *Studio di Tecniche Distribuite di Controllo Predittivo Stocastico per l’Inseguimento di Segnali di Riferimento e Applicazione al Coordinamento di Robot Mobili*. Master’s thesis. 2014.
- [24] T.Kruse et al. “Human-aware robot navigation: A survey”. In: *Robotics and Autonomous Systems, Elsevier* (2013).

- 
- [25] X.Tang, B.Xiao, and K.Li. “Indoor Crowd Density Estimation Through Mobile Smartphone Wi-Fi Probes”. In: *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS* (2018).
  - [26] Y.F.Chen et al. “Socially Aware Motion Planning with Deep Reinforcement Learning”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2017).