CIMAT

Centro de Investigación en Matemáticas, A.C.

ROBOT NAVIGATION IN CROWDS USING QUADRILATERAL VELOCITY OBSTACLES (QVO)

THESIS

Submitted in partial fulfillment of the
requirements for the degree of
Master of Science
with orientation in
Computer Science and Industrial Mathematics
by

Ricardo Andrei Raya Ortega

Thesis Supervisor:
Jean-Bernard Hayet, PhD.
Thesis Co-Supervisor:
Hector Manuel Becerra Fermín, PhD.

Guanajuato, Gto., December 19, 2017

CIMAT Centro de Investigación en Matemáticas, A.C.

ROBOT NAVIGATION IN CROWDS USING QUADRILATERAL VELOCITY OBSTACLES (QVO)

THESIS

Submitted in partial fulfillment of the
requirements for the degree of
Master of Science
with orientation in
Computer Science and Industrial Mathematics
by

Ricardo Andrei Raya Ortega

Thesis Supervisor:
Jean-Bernard Hayet, PhD.
Thesis Co-Supervisor:
Hector Manuel Becerra Fermín, PhD.

Authorization of the final version

Guanajuato, Gto., December 19, 2017

Abstract

This thesis tackles the problem of **robot navigation within crowds**. We propose a reactive methodology, i.e., such that the robot has no full knowledge of the environment and such that its decisions have to be taken in real time. We rely on a well-known geometric approach, the *Velocity Obstacle* approach.

We suppose that we are given a set of trajectory prediction models for a group of mobile obstacles (humans, robots, etc.). The robot takes this information from the environment to go from its initial position to its goal, avoiding collisions. We introduce a conservative discretization of the *VOs*, which we call **Quadrilateral Velocity Obstacle**, *QVO*. It consists of a trapezoid that bounds the *VO* and is computationally easy to model.

We propose two approaches: one Deterministic, and one Probabilistic. The first considers the most probable trajectory for each mobile obstacle. This approach results in an optimization problem with linear constraints in the velocity space. The second approach handles a full trajectory distribution, and decides what velocity to take, both to avoid collisions with obstacles, and to reach its goal. Unlike the deterministic model, this approach contemplates the collision probabilistically.

We present results of the two approaches in simulation, for holonomic and non-holonomic agents, and we provide exhaustive evaluations of the different parameters involved in our algorithms.

I dedicate this thesis to my parents Raúl Raya and Sharys Ortega, to my brothers Raúl, Stephanie and Alain, and to all my family and friends. Thank you for all your support, for your example, for believing in me and supporting me so much.

Acknowledgments

I would like to thank my advisors, Jean-Bernard Hayet, PhD., And Héctor M. Becerra Fermín, PhD., For their help in my master's studies and for accepting me as one of their thesis students. I will never forget your unconditional support and your ability to share your knowledge with me. I also want to thank you because even though you always had a lot of work, you always took time to help me, guide me and sometimes scold me (justifiably), to finish my thesis. Thank you for being so patient and helping me grow academically.

I also want to thank all my teachers, I learned a lot these two and a half years, that in the same way, despite having so much work, they were always willing to resolve my doubts. I also thank CIMAT for giving me everything necessary to achieve this goal, facilities, library, digital resources, etc.

Many thanks also to *Concejo Nacional de Ciencia y Tecnología* (CONACYT), for the financial support that was granted to me for two years, being enough financial support to pursue my master's degree. I also want to thank *Intel Labs* for the scholarship they gave me, and for letting me participate with them in the *Move and let move* project, from which my thesis is derived. And of course many thanks to David Gomez Gutierrez, PhD., and Rafael Eric Murrieta Cid, PhD., for agreeing to be part of the grade dissertation committee, and for carefully reading my thesis draft and provide valuable feedback.

I want to thank my generation colleagues in the same way. Especially I want to thank Ricardo Alonso and Juan Bosco, my roomies, for their support, not only academically, but also personally. Your advice and opinions have been very enriching, you have always listened and helped me. I also thank Vivien, Orly and Mati, for your unconditional friendship and great support.

In the same way, I would like to thank Luis Germán Gutiérrez Torres, PhD (CBTis 217, ITSUR) for his support and initiation in the programming competitions, for presenting me this research environment, and especially, for believing in me.

Of course, special thanks to my family, who has always supported me, who have made me the person I am, for teaching me the values that govern me, for getting me up when I have fallen.

My time at CIMAT was like a range of nuances, many joys, worries, hard work, good rewards. With nostalgia I say goodbye to CIMAT for now, goodbye to this great adventure, I hope not for a long time and return, if destiny so decides, with all the attitude to study a PhD.

Contents

Chapter 1

Theoretical background

1.1 Collision avoidance techniques

There are several categories of collision avoidance methods, namely *single-step* and *multi-step*. In the first category, we usually have a global notion of the workspace, i.e., the dimensions and characteristics of the *world* and obstacles and their location are known [1].

Potential Field Methods (PFM) [2] use a physical analogy assuming that the robot is a charged particle that moves in the workspace under the influence of repulsion/attraction force fields. These force fields are generated by taking into account two factors: the proximity to the *Goal* means greater attractive force, and the proximity to an obstacle or border in the *world* generates greater repulsive force. The *Potential Field* is the sum of both forces. It is generally used in environments without mobile obstacles, although it can be extended to deal with them by converting the method into a reactive approach (multi-step).

Multi-step methods may or may not need global knowledge of the workspace. They are characterized by being more *local and reactive*; that is, they focus on solving the collision avoidance problem constantly (at every step) usually only around the robot [1]. An advantage they have over one-step methods is that they can avoid obstacles discovered in real time because they constantly update their status and relative positions of the obstacles in their *world*. This is very important because it supposes a more realistic performance and is able to face situations that single-step models could not solve. Some of these methods are mentioned below.

Vector Field Histogram (VFH) [3] focuses on estimating a series of candidate motion directions to subsequently select one of them. Initially, the space around the robot is divided into polar segments to be represented by a histogram. The histogram represents the probability that there is an obstacle in each segment, weighted by its likely distance.

The segment chosen from the histogram valley, with density lower than some *threshold*, is the one closest to the segment in the direction towards the *Goal*. The choice of the appropriate velocity can be made by discretizing the selected segment into cells with values that prioritize the distance to the *Goal* and the *clearance* to the obstacles.

Another *multi-step* method is the **Obstacle Restriction Method (ORM)** [4], which is geometrically-based and has been shown to perform an effective movement in dense, complex and cluttered scenarios. It consists of calculating, if necessary, a *sub-Goal* that takes the robot to a region of the space with better possibilities to reach the *Goal*, avoiding deadlocks. To reach the *Goal*, priority is given to reach the *sub-Goal* by restricting the set of velocities that do not move the robot to it even if it means moving away from the *Goal*.

There are also some methods that can complement others, rather than being a method by itself. Examples of these are **Nearness Diagram Navigation (ND)** [5] and **Dynamic Window Approach (DWA)** [6]. The first is a geometrically-based method and consists of applying different behaviors according to specific situations. These situations are based on criteria such as the distance to the obstacles, the *Goal* and/or *secure areas* of the space. Also the current velocity and dynamic constraints of the robot are taken into account. It has been shown that the ND achieves safe robot navigation in complex scenarios.

Dynamic Window Approach consists of generating a *window of attainable velocities* according to the dynamics of the robot. To choose a suitable velocity, a function is evaluated within the window of attainable velocities. The function must take into account the proximity to the *Goal*; this factor leads the robot to its destination. It is also important to prioritize the selection of velocities that move the robot away from obstacles and that have a magnitude close to a desired speed.

Velocity Obstacles (VO) [7] is another geometrically-based, multi-step method consisting of generating an obstacle within the velocity space by combining the dimensions of both the robot and the current obstacle and their velocities. This approach is very practical because it does not require much computational power for its development. We chose *VOs* as a fundamental part of this thesis because it has been proven that, with some modifications, the method can be applied satisfactorily in *multiple-agents* environments [8]. Also, it is not necessary to know the workspace, and the method is robust to the addition of obstacles in real time. This approach and its most important variants are shown in more detail below [9] [10] [8] [11].

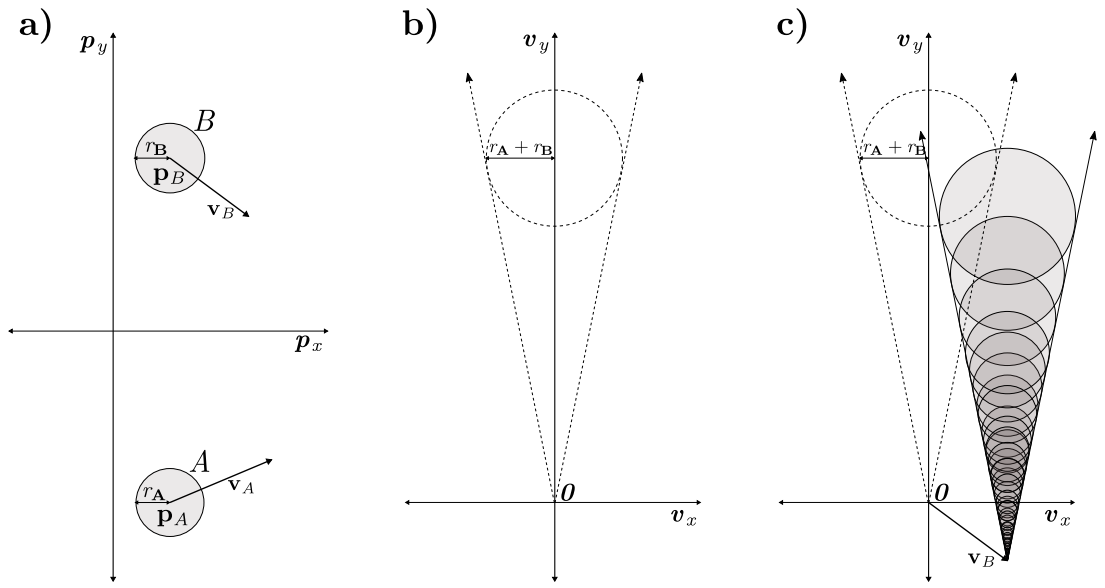


Figure 1.1: **a)** Two disc-shaped agents A and B, with their corresponding radius, positions and velocities. **b)** Collision Cone $CC_{A|B}$ of A induced by B, centered in $\mathbf{p}_A$. **c)** Velocity obstacle $VO_{A|B}$ of A induced by B as an infinite union of circles, the same as $CC_{A|B}$ but moved by $\mathbf{v}_B$ from its apex.

1.2 Velocity Obstacles approach (VO)

A *velocity obstacle VO* [7] is the set of velocities that a moving (controlled) agent could take and that would result in a collision with another obstacle agent, within some time in the future, assuming that both the controlled agent and the obstacle agent maintain their velocity. In other words, if the controlled agent takes a velocity inside the VO and if both agents maintain their velocity, then they will collide; otherwise, it is guaranteed that no collision will occur in the future.

A VO is defined as follows: Let A denote a robot and B a moving obstacle lying in $\mathbb{R}^2$, with circular shapes. Their corresponding radius are r_A, r_B, their current positions are $\mathbf{p}_A, \mathbf{p}_B$, and the velocity of agent B is $\mathbf{v}_B$. The VO of A induced by B is denoted as $VO_{A|B}$, and it represents the set of velocities of A that result in a collision between both agents in some time, assuming that agent B maintains its velocity. If we denote $D(\mathbf{p}, r)$ as a disc of radius r with center $\mathbf{p}$, the $VO_{A|B}$ is given by ***(see Fig. 1.1)***:

$$VO_{A|B} = \{\mathbf{v}|\exists t > 0 :: t(\mathbf{v} - \mathbf{v_B}) \in D(\mathbf{p}_B - \mathbf{p}_A, r_A + r_B)\}.$$

Note that, for one specific t, a collision means:

$$t(\mathbf{v} - \mathbf{v_B}) \in D(\mathbf{p}_B - \mathbf{p}_A, r_A + r_B),$$

which is equivalent to

$$\|\mathbf{p}_B - \mathbf{p}_A - t(\mathbf{v} - \mathbf{v_B})\| \leq r_A + r_B,$$

and to

$$\|\mathbf{v} - (\mathbf{v}_B + \frac{1}{t}(\mathbf{p}_B - \mathbf{p}_A))\| \leq \frac{r_A + r_B}{t}.$$

Seeing $\mathbf{v}$ as a variable in this equation, this means that, for a collision occuring at t, a necessary and sufficient condition is that $\mathbf{v}$ is inside a disc centered at $\mathbf{v}_B + \frac{1}{t}(\mathbf{p}_B - \mathbf{p}_A)$ (which, when t varies, moves along a half-line centered at $\mathbf{v}_B$ with direction $\mathbf{p}_B - \mathbf{p}_A$), with radius $\frac{r_A+r_B}{t}$, inversely proportional to t. Hence, as seen in Fig. 1.1, the collision cone (CC) of the possible relative velocities generating collisions is the union of all these discs when t varies.

The VO corresponds to the collision cone CC moved by $\mathbf{v}_B$, i.e., $CC_{A|B} = VO_{A|B}$ when $\mathbf{v}_B = \mathbf{0}$. The concept of VO is used in this work with some variants. For instance, one of the keypoints of this thesis is that we consider that the moving obstacles are not moving with the same velocities all the time.

Finite-time velocity obstacle

A velocity obstacle with finite-time horizon τ, denoted by $VO^{\tau}_{A|B}$ [12], is defined as the set of velocities of A that result in a collision between the two agents in a time at most τ, if A and B maintain their velocity. This can be expressed as:

$$VO^{\tau}_{A|B} = \{\mathbf{v}|\exists t \in [0, \tau] :: t\mathbf{v} \in D(\mathbf{p}_B - \mathbf{p}_A, r_A + r_B)\}.$$

The velocity obstacle $VO^{\tau}_{A|B}$ can be interpreted geometrically as a truncated cone whose apex is on the origin of the velocity space of the robot A and delimited by the tangents to the disc of radius $r_A + r_B$ centered in $\mathbf{p}_B - \mathbf{p}_A$. The lower bound of the cone corresponds to the arc of circle of radius $(r_A + r_B)/\tau$ centered in $(\mathbf{p_B} - \mathbf{p_A})/\tau$ ***(see Fig. 1.2.a)***. This simple geometric interpretation results from the interpretation of the collision cone seen above, i.e., as the union of discs of radius $\frac{r_A+r_B}{t}$.

A VO bounded between two time instants can be similarly defined as:

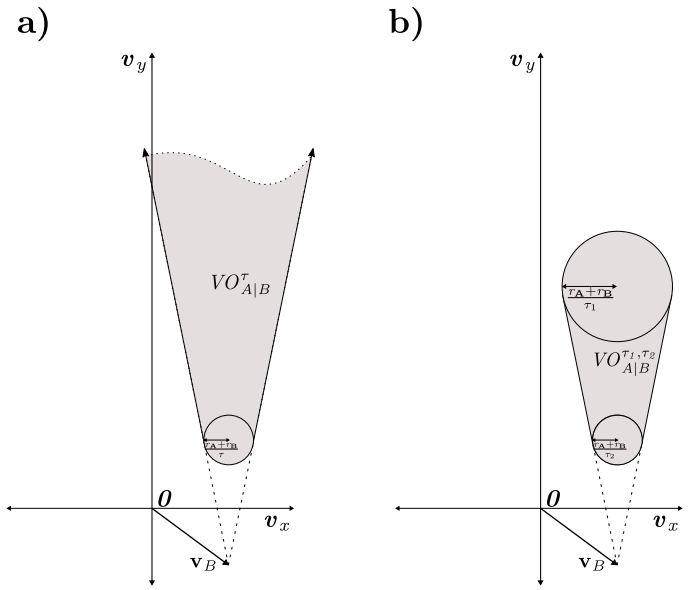


Figure 1.2: **a)** Velocity obstacle $VO^{\tau}_{A|B}$ of A induced by B with time horizon τ. The upper side tends to infinity and corresponds to the set of large velocities in the direction of B, causing instantaneous collision. **b)** Velocity obstacle $VO^{\tau_1,\tau_2}_{A|B}$ bounded between times τ_1 and τ_2.

$$VO^{\tau_1,\tau_2}_{A|B} = \{\mathbf{v} | \exists t \in [\tau_1, \tau_2] :: t\mathbf{v} \in D(\mathbf{p}_B - \mathbf{p}_A, r_A + r_B)\}.$$

The notion of truncated VO will allow to tune the reactivity of the method by neglecting an obstacle for some time if this obstacle is far enough. Thus, the time bounds of the finite-time velocity obstacle can be adapted according to the predicted motion parameters of the obstacles such that the robot does not react immediately nor too late [13]. In the latter case, the robot may not have enough time to avoid the collision.

A few VO variants

The use of *VO* in multi-agent configurations may generate undesired oscillations in paths, as shown in ***Fig. 1.3***. These oscillations are generated in this approach because no robot takes into account that the others also adjust their velocities in each timestep, making velocities that were not valid in the previous step valid again. To address this problem, some approaches have been designed to reduce these oscillations, such as **RVO** [9], **HRVO** [10], **ORCA** [8], **EVO** [11], etc., and they are briefly reviewed below.

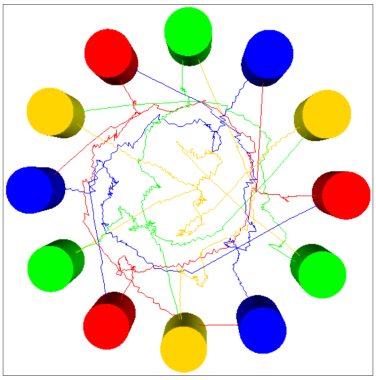


Figure 1.3: Trajectories resulting from 12 robots controlled with the original **VO** approach. Taken from [9]

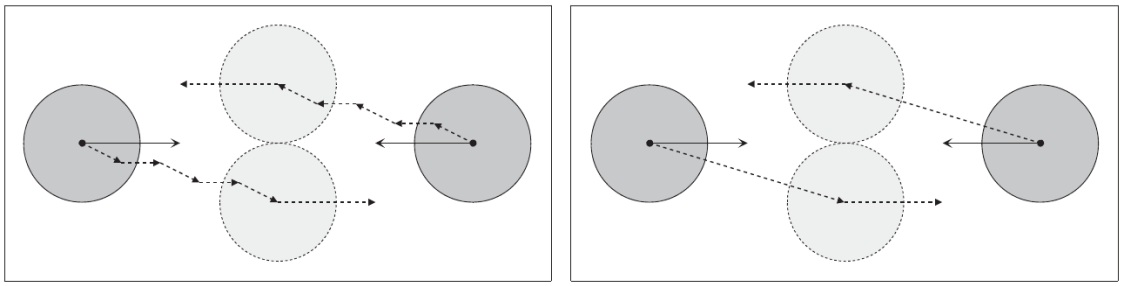


Figure 1.4: Oscillations caused by **VO** (left). Trajectories done when using **RVO** (right). Taken from [9]

RVO and HRVO

In the *Reciprocal Velocity Obstacle* **(RVO)**, it is assumed that in an environment with multiple agents, all agents act similarly, i.e., they all use the same velocity selection algorithm. Collision- and oscillation-free navigation is ensured by mutually (reciprocally) sharing responsibility to avoid a collision, ***(see Fig. 1.4)***. The construction of $RVO_{A|B}$ is similar to $VO_{A|B}$ but with $CC_{A|B}$ moved by $\frac{\mathbf{v}_A+\mathbf{v}_B}{2}$ instead of $\mathbf{v}_B$ ***(see Fig. 1.6 center)***.

When a robot chooses a velocity outside the RVO induced by another robot, both automatically choose to pass through the same side (left or right) of each other, obtaining trajectories free of collisions and oscillations. This approach guarantees the above, as long as there is only one agent influencing another; otherwise, the robots will not necessarily choose the same side to pass. A new problem arises then: *reciprocal dances* ***(see Fig. 1.5 center)***.

In order to solve the problem of *reciprocal dances*, the **Hybrid Reciprocal Velocity Obstacle** *(HRVO)* approach was designed. *HRVO* consists in choosing one of the two sides

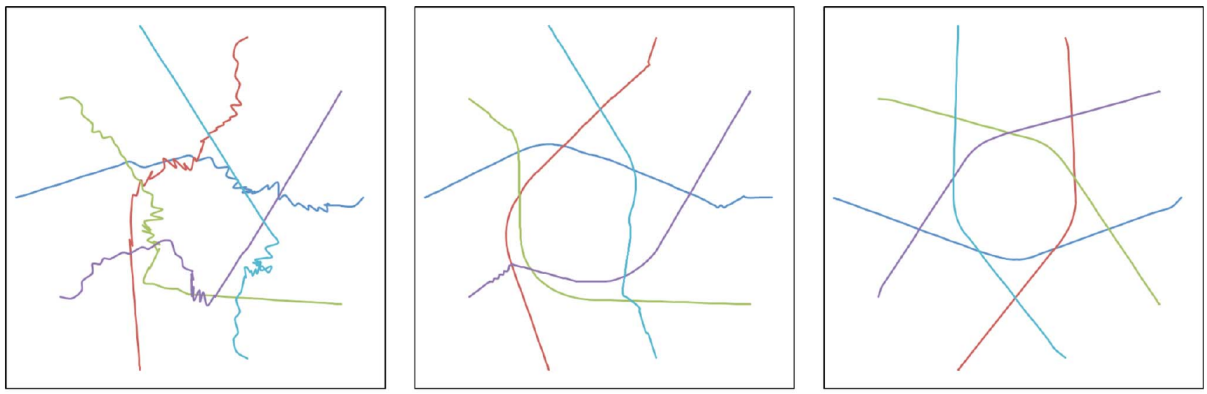


Figure 1.5: Trajectories described by five agents arranged in a circle with their *Goal* on the opposite side. Many oscillations may occur when applying **VO** (left), and reciprocal dances are evident when applying **RVO** (center). When applying **HRVO**, there are no oscillations or reciprocal dances. Taken from [14].

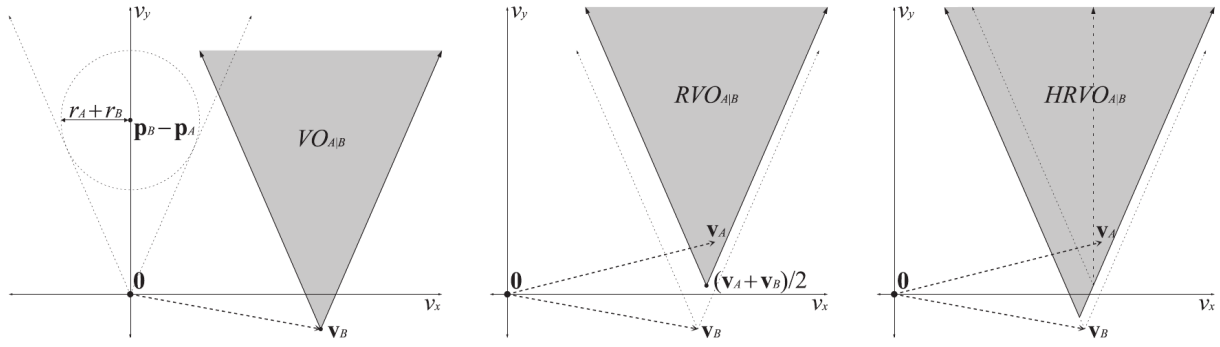


Figure 1.6: Geometric interpretation of $VO_{A|B}$ (left), construction of $RVO_{A|B}$ as a translation of $CC_{A|B}$ at $\frac{\mathbf{v}_A+\mathbf{v}_B}{2}$ (center) and construction of $HRVO_{A|B}$ (right), whose vertex is at the point of intersection of the right side of $RVO_{A|B}$ and the left side of $VO_{A|B}$. Taken from [10].

(left/right) of the other agent for consensus and apply reciprocity. Otherwise, if the other side is chosen, then the agent must take full responsibility to avoid collision. The construction of $HRVO$ is similar to VO and RVO, taking as the vertex the point of intersection between the right (or left) side of RVO with the left side (or right) of VO. ***Fig. 1.5 center and right*** shows trajectories with reciprocal dances generated by applying RVO and trajectories free of oscillations by applying $HRVO$, respectively.

ORCA

In some cases, both **RVO** and **HRVO** produce velocity transitions that may not be smooth. Another disadvantage of these methods is that in very crowded environments, the selection of optimal velocities may be very complex because the space of viable velocities is greatly restricted. To solve this problem, the *Optimal Reciprocal Collision Avoidance* approach [8] was designed to maximize the space of eligible velocities by the use of semi-planes. It *reciprocally* ensures that the robots will be collision-free for at least one time window τ, and it also achieves smooth velocity transitions.

The construction of $ORCA^{\tau}_{A|B}$, read as *ORCA* of robot A induced by obstacle B with a time horizon τ, is defined as follows. Taking into account the $VO^{\tau}_{A|B}$, it is assumed that there will be a collision if $\mathbf{v}_A - \mathbf{v}_B \in VO^{\tau}_{A|B}$. Let $\mathbf{u}$ be the vector from $\mathbf{v}_A - \mathbf{v}_B$ to the nearest point on the edge of the *Velocity Obstacle*:

$$\mathbf{u} = (\underset{\mathbf{v} \in \delta VO^{\tau}_{A|B}}{\operatorname{argmin}} \parallel \mathbf{v} - (\mathbf{v}_A - \mathbf{v}_B) \parallel) - (\mathbf{v}_A - \mathbf{v}_B).$$

Now let $\mathbf{n}$ be the external normal to the edge of $VO^{\tau}_{A|B}$ at the point $(\mathbf{v}_A - \mathbf{v}_B) - \mathbf{u}$. Then $\mathbf{u}$ is the smallest change necessary to avoid collision between the two objects. To assume reciprocity, robot A adjusts its velocity at least in $\frac{\mathbf{u}}{2}$, assuming that B will do the same. More formally:

$$ORCA^{\tau}_{A|B} = \{\mathbf{v} | (\mathbf{v} - (\mathbf{v}_A + \tfrac{\mathbf{u}}{2})) \cdot \mathbf{n} \geq 0\}.$$

Fig. 1.7 shows the geometric representation of $ORCA^{\tau}_{A|B}$. By using semi-planes, the set of possible velocities is restricted for each obstacle. Choosing a velocity outside of these semi-planes ensures collision-free navigation. *ORCA* provides sufficient conditions for the avoidance of obstacles in multiple robot environments using this same strategy.

EVO

A modification of the *VO* approach has been made to handle collision avoidance and oscillations by enlarging the collision cone. *Extended Velocity Obstacle* [11] uses *predicted, communicated, or derived* information about the agents' future intentions. It prevents collisions not only with the current velocity, but also with an interpolation of the future velocity. That way, the robot anticipates the behavior of the other agent and takes actions to obtain a smooth, collision-free and blockage-free trajectory [11]. ***Fig. 1.8*** shows the construction of $EVO_{A|B}$.

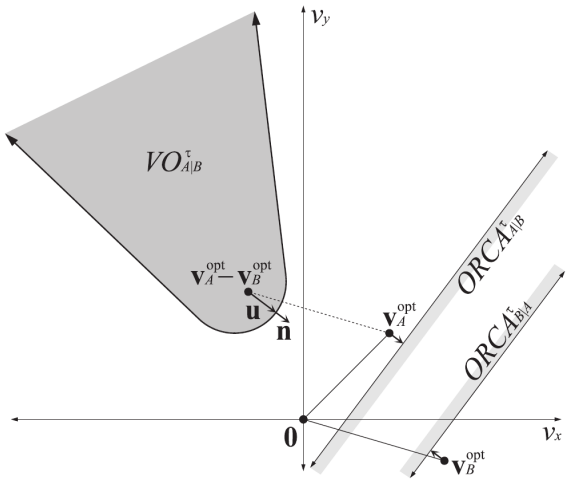


Figure 1.7: The set $ORCA^{\tau}_{A|B}$ of reciprocal velocities permitted for A to achieve collision avoidance with B is a half-plane delimited by the line perpendicular to $\mathbf{u}$ at the point $\mathbf{v}_A + \frac{\mathbf{u}}{2}$. Taken from [8].

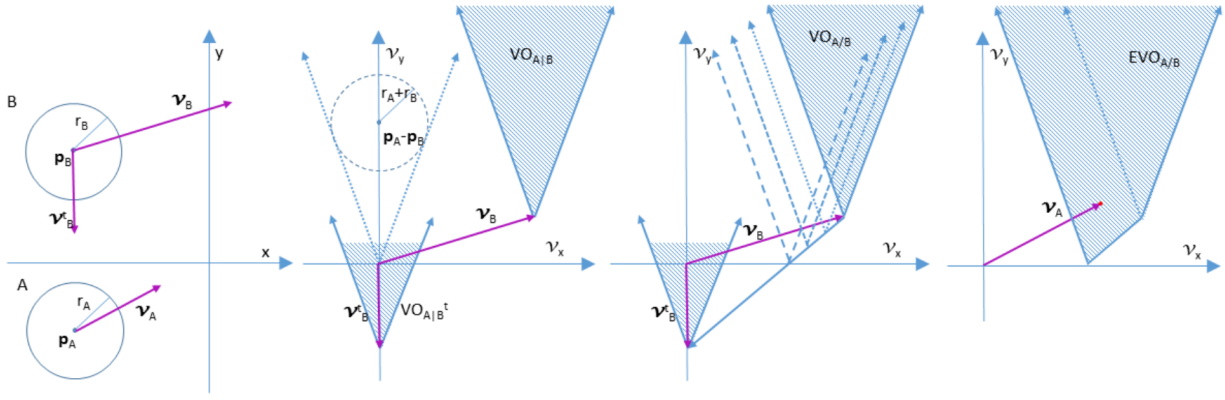


Figure 1.8: Construction of $EVO_{A|B}$. $\mathbf{v}^{\mathbf{t}}_{\mathbf{B}}$ is the *predicted, communicated, or derived* velocity to be reached at some point in time by B. $\mathbf{v}_\gamma$ is the degree of extension, a variable and adjustable parameter. Taken from [11].

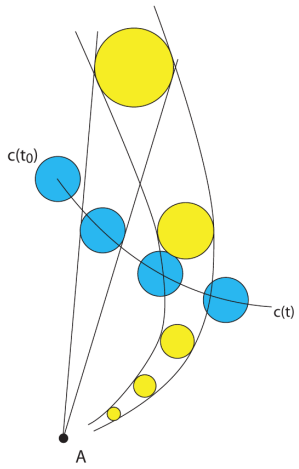


Figure 1.9: In blue, four positions corresponding to different times in which a mobile agent will follow the path $c(t)$. In yellow, five circles representing the edges of the *NLVO* at five different moments of time. The *NLVO* is defined as the infinite sequence of circumferences at each moment of time, represented by velocity obstacles. Taken from [13].

1.3 Non-Linear Velocity Obstacle (NLVO)

Because mobile agents generally follow *nonlinear* trajectories, i.e., arbitrary curves, the conventional *VO* concept is not sufficient to avoid collisions. If the path $c(t)$ that a moving agent will follow is known in advance, it is possible to model the *Nonlinear Velocity Obstacle*, which corresponds to a stretched cone in the velocity space of the robot [15]. If a velocity outside *NLVO* is chosen, collision avoidance is ensured at any point in time as long as the agent follows the trajectory $c(t)$. ***Fig. 1.9*** shows an example of *NLVO*.

Since it is not practical to computationally generate an *NLVO*, *ELVOs* can be used. *Equivalent Linear Velocity Obstacle* consists in taking only a discrete set of times t_i at which the agent moves linearly on the path and in estimating the *VO* to its corresponding piecewise constant velocity, while extrapolating its position according to its current velocity [15].

1.4 Probabilistic Velocity Obstacle

To model the uncertainty that may exist in shape and velocity of mobile obstacles due to the measurement of these parameters, it has been proposed to use **Probabilistic Velocity Obstacles** [16]. *PVOs* mainly start from the notion of *VO*, since the principle is similar in shape and construction.

To handle the uncertainty in shape, the shape is first simplified and is modeled as a circumference for both the robot A_i and for the observed agent A_j, in the same way as for *VO*. The fundamental difference is that the collision is not deterministic (binary) anymore, and real values are obtained in some regions of the **Probabilistic Collision Cone** $PCC_{i,j}$, being a function defined in $PCC_{i,j} : \mathbb{R}^2 \to [0, 1]$.

To represent the uncertainty in shape, two regions corresponding to the sum of the radii of the robot and the obstacle are bounded. The first region is constructed with $inf(r_i + r_j)$, where the density of the probability function is considered to be 1, that is, it is assumed that there will be a collision when choosing a velocity within the VO that it generates. The second region is bounded between $inf(r_i + r_j)$ and $sup(r_i + r_j)$ where the probability density function is considered to be a linear descent function from $inf(r_i + r_j)$ to $sup(r_i + r_j)$, taking values between $(0, 1]$. Both boundaries correspond to the minimum and maximum sum respectively, of the radii r_i and r_j with a high confidence level in the measure of at least 90%. ***See Fig. 1.10***.

Uncertainty in the velocity of A_j must be considered too. As the relationship between CC and VO, the **Probabilistic Velocity Obstacle** PVO is constructed by moving $PCC_{i,j}$ according to velocity v_j, represented as a probability function $V_j : \mathbb{R}^2 \to \mathbb{R}^+_0$, which maps the absolute velocities v_i to its corresponding probability of colliding with A_j.

$$PVO_{i,j}(v_i) = \int_{\mathbb{R}^2} V_j(v_j) PCC_{i,j}(v_i - v_j) d^2 v_j.$$

This corresponds analogously to the convolution of V_j and $PCC_{i,j}$, resulting in

$$PVO_{i,j} = V_j * PCC_{i,j}.$$

In the same way, the *Composed Probabilistic Velocity Obstacle* is defined as the union of all the $PVO_{i,j}$ induced by any obstacle A_j. That corresponds to the probability of not avoiding collisions with such obstacles. Formally it is defined for an agent A_j as the function

$$PVO_i = 1 - \prod_{j \neq i} (1 - PVO_{i,j}).$$

The use of *PVOs* must be implemented differently from the deterministic model to choose the optimal velocity. Unlike the deterministic model that consists of generating *VOs* that restrict the space of velocities in a binary form, i.e., inside the *VOs* one manages the complete probability of a collision, whereas, outside these, the probability of collision is zero. *PVOs* are continuous, which means that there are regions with a greater probability of collision than others. A good algorithm for choosing an optimal velocity under this scheme must first take into account the probability of collision, which should be low or zero, and the accessibility of this velocity.

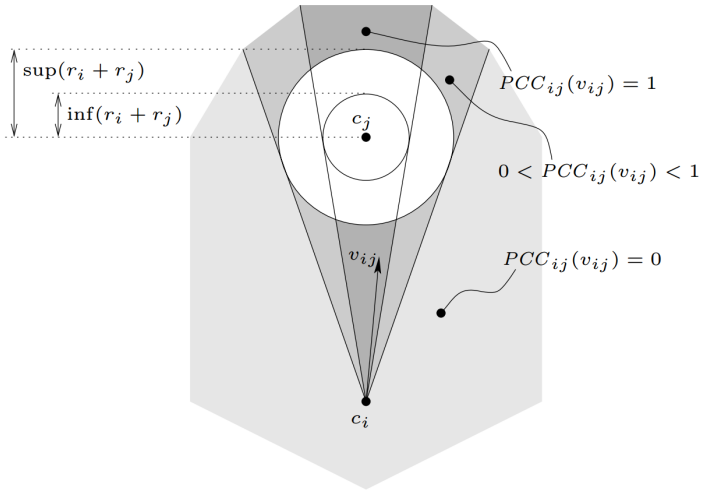


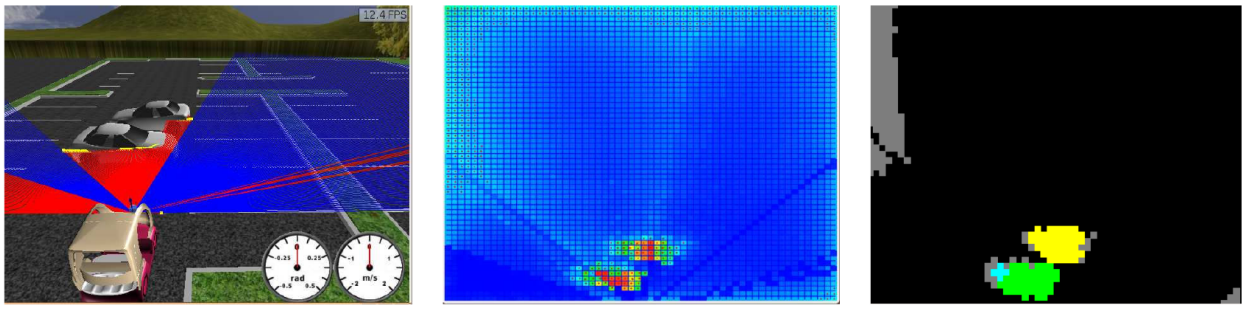
Figure 1.10: Probabilistic Collision Cone. c_i, c_j correspond to the agents' positions i, j. $inf(r_i + r_j)$ is the minimum sum with a high confidence level (usually 90% or higher) of both agents' radii; $sup(r_i + r_j)$ is the maximum sum of of both agents' radii with the same level of confidence.

In [16], a relative utility function is proposed as $RU_i = U_i \cdot D_i \cdot (1 - PVO_i)$, for which it is necessary to choose a velocity v_i that maximizes it. It is not ensured to be collision free, except when $RU_i = 1$ and oscillations will depend on $D_i : \mathbb{R}^2 \rightarrow [0, 1]$. This function describes the accessibility of the new velocity, while $U_i : \mathbb{R}^2 \rightarrow [0, 1]$ is a utility function to navigate to the *Goal*, analogous to $\| v_i - v_i^{pref} \|_2$ in the deterministic model.

On the other hand, other authors have proposed models that combine the *PVOs* with occupation grids in the physical space [17]. This allows to handle obstacles with arbitrary shapes and not just represent them as discs. In [17], *PVOs* are combined with the **Bayesian Occupation Filter** *BOF*. The *BOF* consists in continuously constructing a grid that stores, for each obstacle, a discretized estimate of the *Probabilistic Distribution Function* (pdf) on its velocity and, in the same way, the probability of occupancy of it. Each cell is updated independently; then clusters of cells that share coherent positions and velocities are identified in order to find independent obstacles. ***See Fig. 1.11***.

An advantage of using this type of grid is that the robot makes decisions in a probabilistic context. This allows it to deal with unexpected situations that other models such as *Potential Field Based* or other *Path Planners* cannot handle. To reduce the velocity search space, the grid is modeled according to the robot's kinematic and dynamic constraints.

Additionally, there have been other models that do not use the *PVOs*. Instead, they apply *Probability Density Functions* of collision for the selection of velocities, as is the case



(a) (b) (c)

Figure 1.11: Simulated detection of two cars crossing each other. **(a)** Simulated environment: the robot equipped with a laser range finder detects a car moving from left to right and a second car moving from right to left. **(b)** Dynamic occupancy grid: red is high probability of occupation, blue is low. The space behind the cars has low probability of occupation. **(c)** Clustering: different colours characterise objects and occluded or free space. Taken from [17].

of [18]. Using *pdf's*, one can take into account the uncertainty in volume, position and velocity of the robot and the obstacle. These can be build by approximating with average sampling *(Monte-Carlo)*.

1.5 Motivation of the thesis

As a main motivation, it is desired to control a robot (or eventually a set of robots) in crowded environments, in such a way as to show a behavior free of collisions and oscillations. As input information, we are given a series of trajectory probabilistic models that can be followed by each agent acting as an obstacle *(robot or human)*. Each controlled robot should be able to measure/sense the position of the others agents, and there will no communication between them. In a first approach (called deterministic), we will take the most probable model (trajectory) for each agent and define a *Velocity Obstacle Chain*, defined later. In a second approach, all models with their respective probability will be taken into account to form a weighted sum of collision probability in the velocity space (probabilistic approach), specifically in the *Attainable Cartesian Velocities Region*, which corresponds to the set of dynamically feasible velocities.

The sequel of the document is divided into four main sections. First we dedicate a chapter to the **Deterministic Approach**; as mentioned above, the most likely trajectory model for each agent is used. In the next chapter, we show details about implementation and results of the *Deterministic Modeling*. Then, in another chapter we describe the **Probabilistic Approach**, where we consider a probability distribution of the trajectory for each agent and choose the safest velocity according to the probability density function of colli-

sion. A chapter is also dedicated to the implementation and results of this methodology.

Chapter 2

Deterministic Approach

Our *deterministic modeling* uses only one predicted trajectory for each agent to define the velocity obstacle. It is assumed to be the one that the agent *(moving obstacle)* will follow, and, in case we handle a distribution over trajectories, it may correspond to the statistical mode of the distribution, or its mean. Under these assumptions, the whole process is geometric, and binary, i.e., in this model, *safe* and *conflictive* areas in the velocity space are well-defined. If the robot chooses velocities inside the *safe* area, a collision-free path is completely guaranteed. Otherwise, the robot will necessarily have a collision at some future time instant if it maintains its selected velocity. The aim is to bring the robot to its *Goal* while following a collision-free path, prioritizing the last feature (avoid collision) over the first (reach the goal).

2.1 Quadrilateral based Velocity Obstacle

We recall that $VO^{\tau_1,\tau_2}_{A|B}$ (see Chapter 1) is a velocity obstacle induced by B on A and reduced to a time interval $[\tau_1, \tau_2]$. It is delimited by line segments and arcs of circle. In order to model it in a simpler and computationally more practical manner, we propose to approximate it by the trapezoid of minimum area circumscribed to the corresponding VO. We name this approximation **Quadrilateral Velocity Obstacle** QVO, ***see Fig. 2.1.c***, and it is defined as follows. Consider $S(\mathbf{p}, d)$ the line segment perpendicular to the line joining $\mathbf{p}$ to the origin, and centered on the same point, with dimension $2d$. Then, let us define $\tau_1^- = \frac{\tau_1\|\mathbf{p}_B-\mathbf{p}_A\|}{\|\mathbf{p}_B-\mathbf{p}_A\|+(r_A+r_B)}$ and $\tau_2^+ = \frac{\tau_2\|\mathbf{p}_B-\mathbf{p}_A\|}{\|\mathbf{p}_B-\mathbf{p}_A\|-(r_A+r_B)}$ as the times that correspond to the farthest extreme points of $VO^{\tau_1,\tau_2}_{A|B}$, along its axis of symmetry.

In order to explain more clearly the construction of QVO, we focus on the special case where $\mathbf{v}_B = \mathbf{0}$, and $\tau_1 = 1$. We define θ as the angle with respect to the direction $\mathbf{p}_B - \mathbf{p}_A$ of the external bi-tangents between the circles $D(\mathbf{p}_B - \mathbf{p}_A, r_A + r_B)$ and $D(\frac{\mathbf{p}_B-\mathbf{p}_A}{\tau_2}, \frac{r_A+r_B}{\tau_2})$, as depicted in ***Fig. 2.2***.

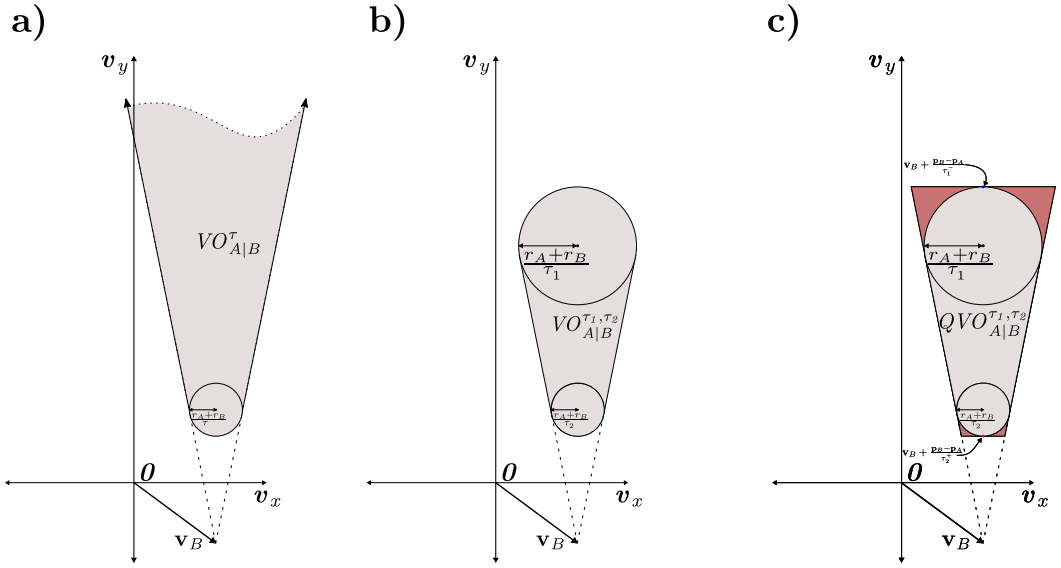


Figure 2.1: Extension of ***Fig.* 1.2** to show the QVO construction. **a)** Velocity obstacle $VO^{\tau}_{A|B}$ for A induced by B with time horizon τ. The upper side tends to infinity and corresponds to the set of large velocities in the direction of B, causing instantaneous collision. **b)** Velocity obstacle $VO^{\tau_1,\tau_2}_{A|B}$ bounded between times τ_1 and τ_2. **c)** Construction of the QVO corresponding to $VO^{\tau_1,\tau_2}_{A|B}$. Red areas are improperly constrained due to the conservative choice for the shape of QVO.

By simple geometric analysis, and with the help of ***Fig. 2.2***, we can deduce:

$$\beta = \arcsin\left(\frac{r_A+r_B}{\|\mathbf{p}_B-\mathbf{p}_A\|}\right),$$

$$\theta = \frac{\pi}{2} + \beta,$$

$$c = (r_A + r_B)\cos\theta.$$

Then, we define $cr_{A,B}$ as the *composed radius* of A and B. It corresponds to the distance between $\mathbf{p}_B - \mathbf{p}_A$ and the point on the bitangent. Furthermore, we define $b = \|\mathbf{p}_B - \mathbf{p}_A\|$. Since the green and yellow triangles in ***Fig. 2.2*** are *similar*, we have $a/b = a'/b'$, where $b' = b - c$. It becomes straightforward to deduce

$$cr_{A,B} = a = \frac{(r_A+r_B)\|\mathbf{p}_B-\mathbf{p}_A\|}{\|\mathbf{p}_B-\mathbf{p}_A\|-c}.$$

Once we have found $cr_{A,B}$, we can formally define $QVO^{\tau_1,\tau_2}_{A|B}$ similarly to $VO^{\tau_1,\tau_2}_{A|B}$, as follows:

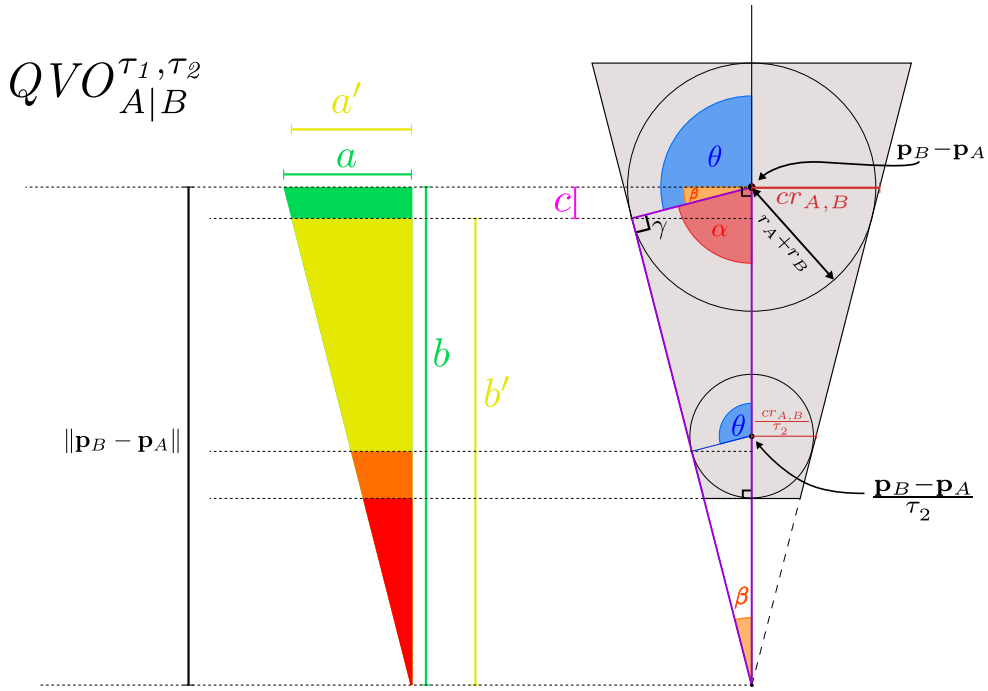


Figure 2.2: Construction of the QVO.

$$QVO^{\tau_1,\tau_2}_{A|B} = \{\mathbf{v} | \exists t \in [\tau_1^-, \tau_2^+] :: t\mathbf{v} \in S(\mathbf{p}_B - \mathbf{p}_A, cr_{A,B})\}.$$

In this way, we obtain a quadrilateron as an approximation of the VO, and with which it is easier to evaluate whether a candidate for the optimal velocity, $\mathbf{v}_A^{opt}$, is inside or outside the shape (mentioned later as a part of the algorithm), as it consists only on tests on semi-planes. Moreover, it only requires storing four points in $2D$. Otherwise, it would be necessary to store only two points corresponding to the center of each of the circles delimiting the $VO^{\tau_1,\tau_2}_{A|B}$ at their ends, and their respective radii. Although this would give greater precision when defining the safe and conflicting areas and would use less memory, it becomes more complex to validate if a candidate for the optimal velocity $\mathbf{v}_A^{opt}$ is inside or outside the VO, due to the curvature of its ends.

VO Chains

The aforementioned QVO model is also useful for creating a *Velocity Obstacle Chain*, $VOC_{A|B}$ for A, induced by a mobile agent B, from which a trajectory has been predicted or is known. This predicted trajectory is discretized and is seen as the union of *chords* (line segments) between some points on it. Each sub-path *(chord)* corresponds to the velocity and position of the obstacle along different periods of time, ***see Fig. 2.3***. In this approxi-

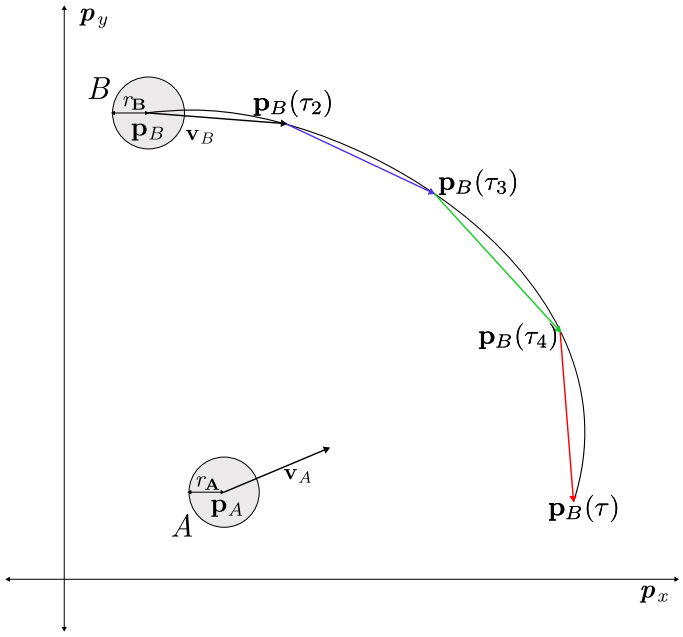


Figure 2.3: Discretization of a trajectory in chords to simplify the predicted behavior of the mobile agent. Each $\mathbf{p}_B(\tau_i)$ corresponds to the position predicted for $\mathbf{p}_B$ at each selected point on the chord.

mation, the velocity is piecewise constant, on each sub-path.

More specifically, each agent *(mobile obstacle)* is associated with a continuous probability distribution that predicts its trajectory for a few time steps in the future; examples of these probability distributions will be given in the next chapters. One of the kind of models we handle is a *mixture* of simpler models (e.g., Gaussian Processes). Depending on the predictor and observed data, these models are regularly updated, and we can deduce a deterministic predicted trajectory within the next time span τ, by using the mean or the mode of this distribution.

For simplicity, it is assumed that the trajectory is traced with a piecewise constant speed by the agent, and that the *n* points taken from it form *n-1* chords which are equidistant in time. It is also assumed that $n > 2$ and that both endpoints corresponding to $\mathbf{p}_B(\tau_1)$ and $\mathbf{p}_B(\tau_n)$ are used.

Formally, the *VOC* (*Velocity Obstacle Chain*) for A induced by B, for a time horizon τ, is defined as:

$$VOC^{\tau}_{A|B} = \bigcup_{i=1}^{n-1}\{\mathbf{v}|\exists t \in [\tau_i, \tau_{i+1}] :: t(\mathbf{v} - \mathbf{v}_B(\tau_i)) \in D(\mathbf{p}_B(\tau_i) - \mathbf{v}_B(\tau_i)\tau_i - \mathbf{p}_A, r_A + r_B)\},$$

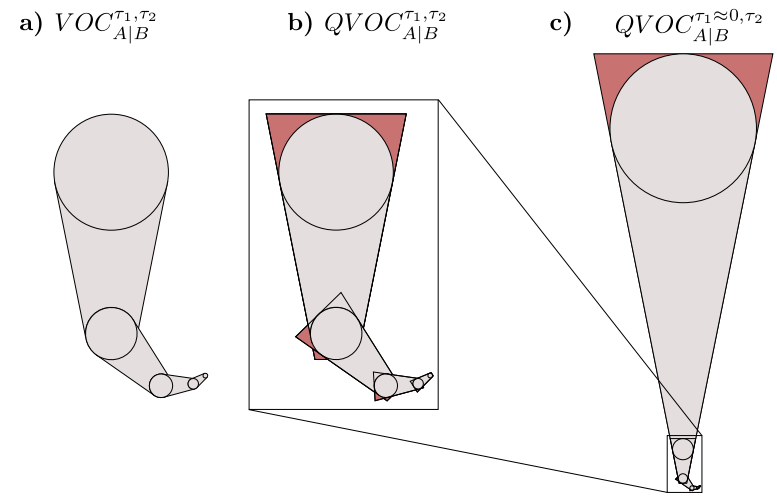


Figure 2.4: **a)** Theoretical shape of $VOC^{\tau_1,\tau_2}_{A|B}$. **b)** Shape of $QVOC^{\tau_1,\tau_2}_{A|B}$, where red areas are improperly constrained because of the QVO model. **c)** Shape of $QVOC^{\tau_1\approx 0,\tau_2}_{A|B}$ when the lower time is approximated to 0, the area that is improperly constrained is of little significance.

where n is the number of sample points taken from the trajectory which divide it in $n-1$ chords, $[\tau_i, \tau_{i+1}]$ is the *time window* corresponding to each chord, starting with $\tau_\mathbf{1} = 0$ and finishing with $\tau_\mathbf{n} = \tau$, and $\mathbf{v}_B(\tau_i)$ is the associated (constant) velocity of B with which it will trace the chord. Then, $VOC^{\tau}_{A|B}$ corresponds to the union of every single $VO^{\tau_i,\tau_{i+1}}_{A|B}$ within τ. Note that this is a generalization of the concept of Velocity Obstacles for time-varying velocities of the moving obstacle, where, for simplicity, the evolution of these velocities is assumed piecewise constant. Then, we define a $QVOC^{\tau}_{A|B}$, *Quadrilateral Velocity Obstacle Chain within* $[0,\tau]$ *of* A*, induced by* B as follows:

$$QVOC^{\tau}_{A|B} = \bigcup_{i=1}^{n-1}\{\mathbf{v}|\exists t \in [\tau_i,\tau_{i+1}] :: t(\mathbf{v}-\mathbf{v}_B(\tau_i)) \in S(\mathbf{p}_B(\tau_i)-\mathbf{v}_B(\tau_i)\tau_i-\mathbf{p}_A, cr_{A,B})\}.$$

It would seem that with this modeling, the set of collision-free velocities is excessively restricted, since when inscribing each VO inside a trapezoid (QVO) there are areas that are improperly constrained. In ***Fig. 2.1 c***, we see in red these improperly constrained areas due to the conservative shape of QVO, but in ***Fig. 2.4 b*** and ***c***, we can see that much of the red zone of a QVO is part of the region corresponding to the previous segment and thanks to recalculation in each iteration, the loss is of little significance if the trajectory curvature is not too high.

2.2 Selection of the optimal velocity

Recall that we intend to bring a robot A to its goal $\mathbf{p}_A^{Goal}$, which consists of a point on the Cartesian plane. The robot A has, as intrinsic parameters, the magnitudes s_A^{pref} and s_A^{max} corresponding to its preferred and maximum speeds, i.e., the magnitude of the velocity at which it is preferred to control the robot most of the time and the maximum speed magnitude it is able to reach. The robot A must reach its goal without causing collisions with other mobile agents or fixed obstacles.

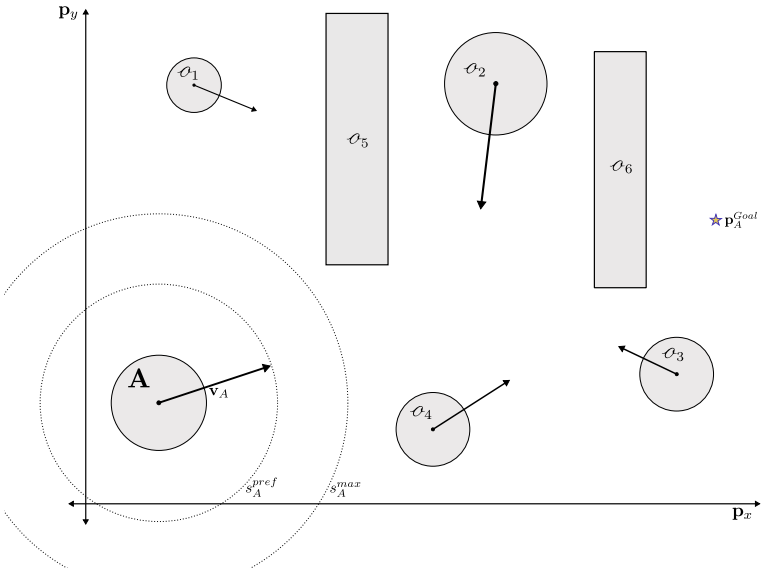


Figure 2.5: A robot A in a dynamic environment with fixed obstacles $\mathcal{O}_5$ and $\mathcal{O}_6$ and mobile obstacles $\mathcal{O}_1, \ldots, \mathcal{O}_4$, aims to reach its goal $\mathbf{p}_A^{Goal}$ while avoiding collisions, using velocities of magnitude s_A^{pref} (inner circle) and remaining at the margin of maximum speed s_A^{max} (outer circle).

In the previous section, we have defined the $VOC_{A|B}^{\tau}$ for only one robot A and one mobile obstacle B, with a predicted trajectory for B, within the time interval $[0, \tau]$ starting from the current instant. Now suppose that we have a set $\mathcal{O}$ of obstacles (mobile and fixed). We define the *combined* VO for A, denoted as VO_A^{τ}, as the union of all $VO_{A|\mathcal{O}_j}^{\tau_i,\tau_{i+1}}$ within $[0, \tau]$, induced by each obstacle $\mathcal{O}_j$ in $\mathcal{O}$, as follows:

$$VO_A^{\tau} = \bigcup_{\mathcal{O}_j \in \mathcal{O}} VOC_{A|\mathcal{O}_j}^{\tau}.$$

In order to choose an *optimal velocity*, we first need to compute the set of ***Attainable Cartesian Velocities*** (ACV)[13] that A could reach in a time lapse Δt, i.e., the physically feasible velocities. We define $\mathcal{U}_A$ as the set of permissible controls for A. For the moment,

we are focusing only in holonomic robots, hence the control $\mathbf{u}$ is defined as the instantaneous velocities along each axis:

$$\mathbf{u} = \begin{bmatrix} v_x \\ v_y \end{bmatrix},$$

with v_x and v_y the velocities along each direction in the Cartesian plane.

The set $\mathcal{U}_A$ takes into account the dynamic and kinematic constraints of the robot. First, we define $\dot{\mathbf{u}}$ as the instantaneous acceleration taken by A to reach the optimal velocity. When implemented, this term is simply approximated by the difference between the selected optimal velocity and the current one, divided by the time lapse Δt between consecutive velocity decisions. In each direction, this acceleration is bounded between the (positive) maximum reachable acceleration of the robot, $\dot{\mathbf{u}}_A^{Max}$, and its (negative) maximum reachable deceleration $\dot{\mathbf{u}}_A^{max}$.

Second, the magnitude of $\mathbf{u}$ is at most s_A^{max}, the maximum reachable speed of A. By combining the velocity and acceleration constraints, $\mathcal{U}_A$ is defined as follows:

$$\mathcal{U}_A = \{\mathbf{u} \in \mathbb{R}^2 \mid \|\mathbf{u}\| \leq s_A^{max} :: \dot{\mathbf{u}}_{A_x}^{max} \leq \dot{v}_x \leq \dot{\mathbf{u}}_{A_x}^{Max}; \dot{\mathbf{u}}_{A_y}^{max} \leq \dot{v}_y \leq \dot{\mathbf{u}}_{A_y}^{Max}\}.$$

Once we calculate $\mathcal{U}_A$, we can now define the set of *Attainable Cartesian Velocities* of A, $ACV_A(t)$, by integrating the admissible controls $\mathbf{u} \in \mathcal{U}_A$ from the current state $(\mathbf{p}_A, \mathbf{v}_A(t))$. For holonomic robots, it is defined as:

$$ACV_A(t) = \mathcal{U}_A.$$

To solve the problem that we have formulated initially (reach a goal and simultaneously avoid static and dynamic obstacles), we must find a velocity $\mathbf{v}_A^{opt}$ for each time step Δt, which corresponds to a point in the velocity space such that its distance to $\mathbf{v}_A^{pref}$ is minimal and which does not cause a future collision, i.e., a velocity which is outside VO_A^τ, subject to the velocity and acceleration constraints of A. More formally, the optimization problem is defined as:

$$\mathbf{v}_A^{opt} = \underset{\mathbf{v} \in ACV_A(t) \setminus VO_A^\tau}{\operatorname{argmin}} \| \mathbf{v} - \mathbf{v}_A^{pref} \|,$$

where $ACV_A(t) \setminus VO_A^\tau$ is the set of velocities within $ACV_A(t)$ but not in VO_A^τ, as shown in ***Fig. 2.7***. This velocity is applied to the robot on time intervals of duration Δt, and is then re-estimated again. Note that the problem stated in this way is a convex optimization problem with linear inequalities (VO and control constraints).

One option to solve it is the *ClearPath* algorithm, presented in [12]. We propose an alternative to it, starting from the following lemma, where $\mathcal{BE}$ (for *Boundary Edges*) is the set of line segments that delimit the contour of VO_A^τ.

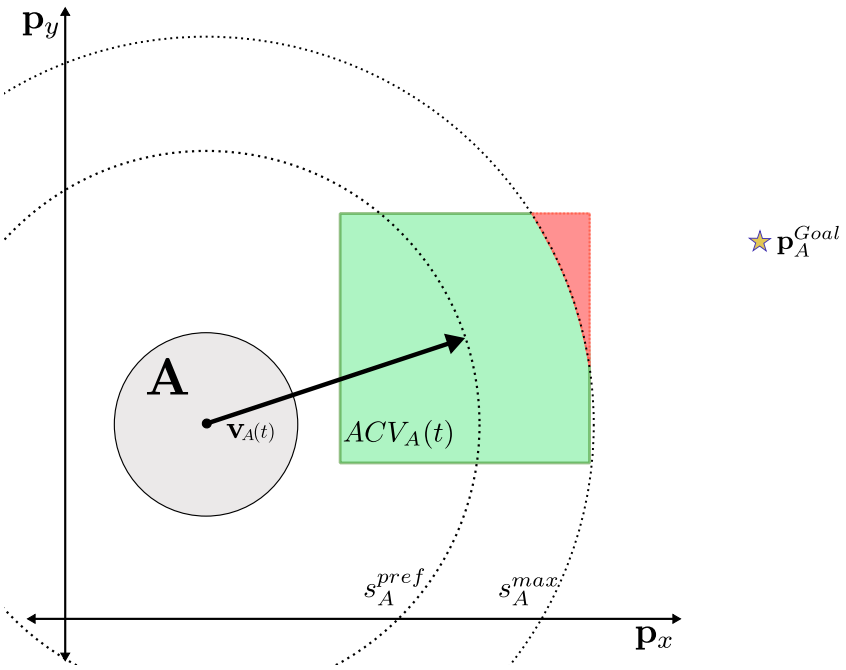


Figure 2.6: Restrictions on velocity space of A. The arrow points to the current velocity $\mathbf{v}_A(t)$ at t. The square around $\mathbf{v}_A(t)$ corresponds to the acceleration constraints (in this case $\dot{\mathbf{u}}_{A_x}^{max} = -\dot{\mathbf{u}}_{A_x}^{Max}$, same as $\dot{\mathbf{u}}_{A_y}^{max} = -\dot{\mathbf{u}}_{A_y}^{Max}$). The green area corresponds to $ACV_A(t)$, and the red area is the set of controls invalidated due to maximum speed constraints (dashed external circle).

Lemma 2.2.1. *When velocities are unconstrained, if $\mathbf{v}_A^{pref}$ is inside VO_A^{τ}, then $\mathbf{v}_A^{opt}$ must lie on $\mathcal{BE}$.*

Proof. We prove it by contradiction. Let $\mathbf{v}_A^{opt}$ not be on one of the edges $\mathcal{BE}$ and say it minimizes the distance from $\mathbf{v}_A^{pref}$. Consider a disk of radius $\epsilon \left|\mathbf{v}_A^{opt} - \mathbf{v}_A^{pref}\right|$, centered at $\mathbf{v}_A^{opt}$, with $\epsilon > 0$. For a sufficiently small ϵ, the circle is completely outside VO_A^{τ}, and we have a point $\mathbf{v}_A^{opt} + \epsilon(\mathbf{v}_A^{pref} - \mathbf{v}_A^{opt})$ that is outside and at a distance of $(1-\epsilon)\left|\mathbf{v}_A^{opt} - \mathbf{v}_A^{pref}\right|$ from $\mathbf{v}_A^{pref}$. Thus we arrive at a contradiction as we could exhibit a better optimal point. Therefore, $\mathbf{v}_A^{opt}$ must lie on the boundary segment of one of the VOs [12]. □

To complement and follow the reasoning of ***lemma 2.2.1***, we consider the case with constraints on velocity and formulate the following, similar lemma:

Lemma 2.2.2. *In the case with velocity constraints, if $\mathbf{v}_A^{pref}$ is outside $ACV_A(t)$, $\mathbf{v}_A^{opt}$ must lie on $\mathcal{BE}$ and/or $\mathcal{B}ACV_A(t)$.*

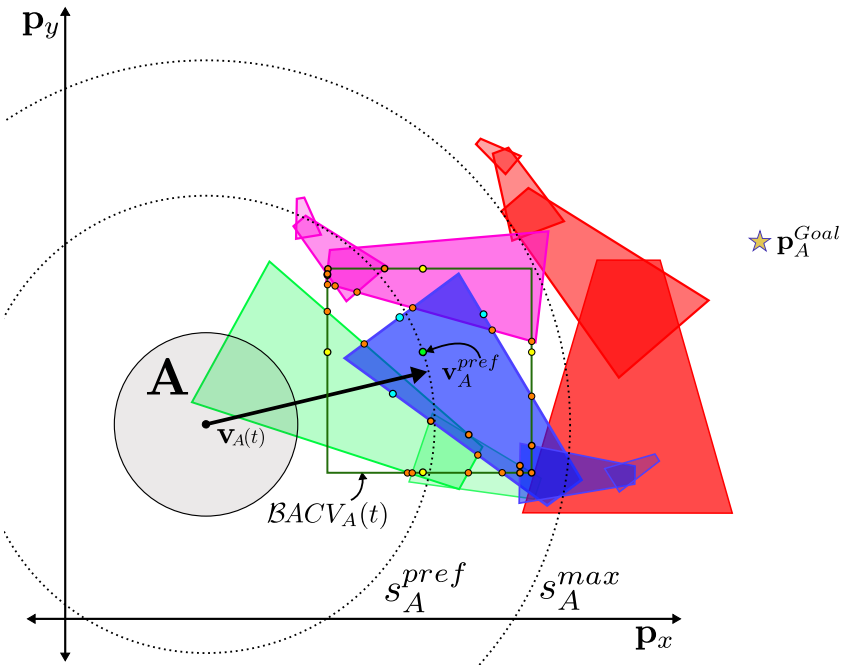


Figure 2.7: In red, magenta, blue and green colors, the $QVOs$ induced on A by four mobile obstacles whose union gives rise to VO_A^τ. $ACV_A(t)$ is shown as the dark green outline square, corresponding to a holonomic robot. In this configuration, $\mathbf{v}_A^{pref}$ (green dot) is inside the first set. We want to find $\mathbf{v}_A^{opt}$ within $ACV_A(t) \backslash VO_A^\tau$; to achieve this, we calculate a series of candidates within $ACV_A(t)$. Nonviable candidates are represented as points with thicker contour.

Proof. As proof of this lemma we consider the union of the complement of $ACV_A(t)$ and VO_A^τ, and $\mathcal{B}ACV_A(t)$, the boundary of $ACV_A(t)$. Then we can use exactly the same argument from the proof of ***lemma 2.2.1***. □

Once we have proved than $\mathbf{v}_A^{opt}$ must lie on $\mathcal{BE}$ and/or $\mathcal{B}ACV_A(t)$ when $\mathbf{v}_A^{pref}$ is not valid, we can go even further and show that we can consider only a finite set of *candidate* points for $\mathbf{v}_A^{opt}$. Since the initial search space is infinite, this leads us to formulate the following lemma:

Lemma 2.2.3. *If $\mathbf{v}_A^{pref}$ is in conflict by violating either collision or feasibility constraint, then $\mathbf{v}_A^{opt}$ must lie at some of the intersections of the boundaries of the VOs ($\mathcal{BE}$); or at some intersection of $\mathcal{BE}$ with $\mathcal{B}ACV_A(t)$; or at some of the orthogonal projections of $\mathbf{v}_A^{pref}$ on the boundary of some VO that contains it; or finally at some of the orthogonal projections of $\mathbf{v}_A^{pref}$ on $\mathcal{B}ACV_A(t)$.*

Proof. Consider Fig. 2.8. We prove this lemma by first defining $\mathcal{VB}$ as the set of valid boundaries of $\mathcal{B}ACV_A(t)$ and $\mathcal{BE}$, i.e., such that none of their line segments are in conflict. Then, we arbitrarily take one point $\mathbf{h}$, different from the points described in the lemma,

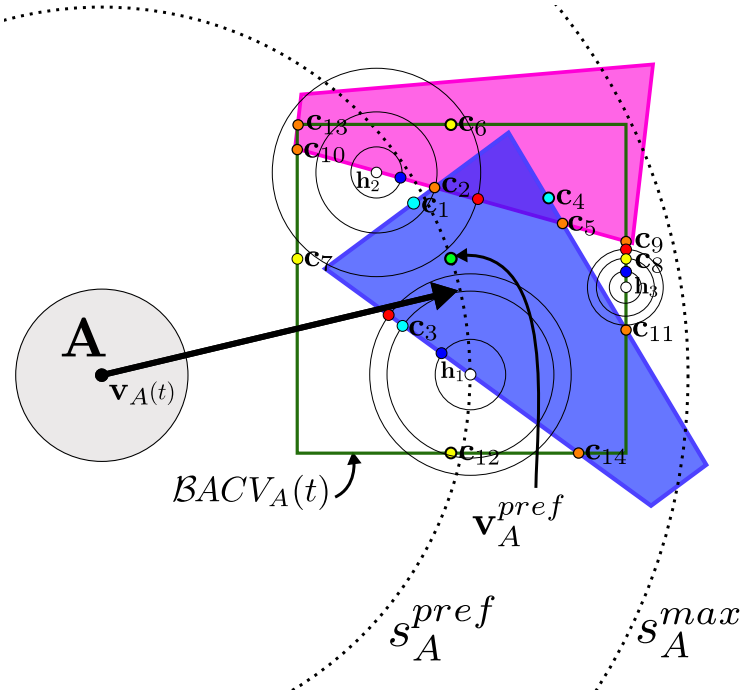


Figure 2.8: In magenta and blue, two $QVOs$ induced on A by two obstacles. $ACV_A(t)$ is shown as the dark green outline square, corresponding to a holonomic robot. The the candidates found using ***lemma 2.2.3*** are named as $\mathbf{c}_i$. Arbitrary points along $\mathcal{VB}$ are shown as $\mathbf{h}_j$. In blue and unlabeled, points closer to $\mathbf{v}_A^{pref}$ than $\mathbf{h}_j$. In red and unlabeled, points farther to $\mathbf{v}_A^{pref}$ than $\mathbf{c}_i$. In this example, we show that $\mathbf{c}_3$, $\mathbf{c}_2$ and $\mathbf{c}_8$ are closer points to $\mathbf{v}_A^{pref}$ than $\mathbf{h}_1$, $\mathbf{h}_2$ and $\mathbf{h}_3$, respectively.

from the infinite set of points along one line segment $\mathbf{vb}_i$ of $\mathcal{VB}$. If we consider a disc d_h centered on $\mathbf{h}$ with a radius $\epsilon > 0$ sufficiently small so that the two intersections of the boundary of d_h and $\mathbf{vb}_i$ lie on the latter, then one point $\mathbf{q}$, of the two new generated, is closer to $\mathbf{v}_A^{pref}$ than $\mathbf{h}$, except for one of the points $\mathbf{c}$ described in the lemma. This results from the convexity of the minimized function. Then we would have found a better candidate, which is a contradiction. □

Taking these lemmas as a reference, a set $\mathcal{C}$ of candidates to choose $\mathbf{v}_A^{opt}$ can be created as all points, represented in orange in ***Fig. 2.7***, corresponding to the intersections between $\mathcal{B}ACV_A(t)$ and all the VOs that share area with it and the intersections between these VOs. Candidates are also added to $\mathcal{C}$ as a result of projecting $\mathbf{v}_A^{pref}$ on the edges of all VOs containing it as it is done in [14], and as it is shown in the figure in cyan, and finally the projections of $\mathbf{v}_A^{pref}$ on $\mathcal{B}ACV_A(t)$, which are shown in yellow. Even though many of the candidates will not be on $\mathcal{VB}$ (nonviable), as represented as points with thicker contour in ***Fig. 2.7***, they can become viable when constraints are relaxed (i.e., when we remove some of them, as described below).

Algorithm for selecting the optimal velocity

We will now describe the ***Algorithm 1*** (see below) for selection of the optimal velocity of the robot. As an input, a set of parameters is received, as described hereafter:

τ: the *time window* in which obstacle trajectories are known/predicted.

$\Delta\tau$: *relaxation time*, is the time span by which τ is reduced when restrictions on time are relaxed.

τ_{min}: *minimum time*, is the minimum time span by which τ can be reduced, i.e., $\tau \geq \tau_{min}$.

nO: higher value of the maximum number of obstacles that the robot can *observe*, that is, for which the robot can estimate their positions and velocities. This value is set depending on the memory and computational capacity available.

nmO: lower value of the maximum number of obstacles that the robot can *observe*. This second value is set when relaxing the restrictions when no optimal velocity $\mathbf{v}_A^{opt}$ has been found. Typically, $nO \geq nmO$.

$\mathcal{O}$: set of obstacles visible by the robot, where each $o \in \mathcal{O}$ has its own trajectory prediction models within a time horizon τ_o.

In order to improve the performance of the algorithm, we can discard the VOs that do not influence the robot within the future time span $t + t_c$, where t_c is the maximal *time to collision* that we want to take into account. Then we define $ACV_A^{view}(t_c)$ as an area, in the velocity space, corresponding to all the achievable velocities within times $[t, t + t_c]$, i.e., it is an expanded version of the $ACV_A(t)$.

The first 7 lines of Algorithm 1 consist of calculating $\mathcal{LVO}$, i.e., the list of VOs induced by the nO closest obstacles (or less if there is not that many) that intersect with $ACV_A(t)$, and sort them by t_{start} (the lower bound of time for each VO). We also verify that $\mathbf{v}_A^{pref}$ is not in conflict. If the latter is out of $\mathcal{LVO}$ and, at the same time, is reachable (is inside $ACV_A(t)$), then $\mathbf{v}_A^{opt}$ will be $\mathbf{v}_A^{pref}$.

In case v_{pref} is in conflict, the instructions will be executed until line 13. These consist of generating $\mathcal{C}$, the set of candidates described by ***lemma 2.2.3***, and choosing the best among them. To do so, we iterate among them in ascending order by their distance to $\mathbf{v}_A^{pref}$, and in case we found a viable candidate, then $\mathbf{v}_A^{opt}$ will be the best of them, c.

If we have not found $\mathbf{v}_A^{opt}$ at this time, we relax the constraints, first by reducing the time window τ (i.e., by not considering potential collisions far in the future), and then, if necessary, by reducing the number of obstacles visible by the robot. If we reduce the constraints, we may get a smaller $\mathcal{LVO}$, and the velocity space may be less restricted. Thus,

the number of candidates in $\mathcal{C}$ decreases and some of them may now be viable.

In lines 14 to 18, we iteratively relax the constraints to find $\mathbf{v}_A^{opt}$, by reducing τ by increments $\Delta\tau$ in each step while $\tau \geq \tau_{min}$. By doing this, $\mathcal{LVO}$ is now the set of VOs for which t_{start} (lower time limit) is less than τ.

The rest of the algorithm consists of relaxing the velocity restrictions by *ignoring* some obstacles, i.e., we now focus only on up to nmO obstacles. Note that a possible implementation could consider different priorities among obstacles, i.e., we could give high priority to obstacles that should be avoided at any cost. After updating the candidates in $\mathcal{C}$, we look again for $\mathbf{v}_A^{opt}$ among them. In case of not finding a viable candidate, we gradually reduce the velocity until reaching a zero velocity (i.e., stopping the robot), hoping that in future iterations, the restrictions will be solved and the dynamic obstacles will try to avoid the robot, so we give higher priority to avoid collisions than to reach the robot's *Goal*.

Handling non-instantaneous velocity changes

In the previous paragraphs, we have assumed implicitly that the desired velocity $\mathbf{v}_A^{opt}$ can be reached instantaneously at time t, when actually it is the case only after some delay, i.e., at $t + \Delta t$. This produces a position error e in $\mathbf{p}_A(t + \Delta t)$, that can be bounded as:

$$\begin{aligned} e &= \|\textstyle\int_t^{t+\Delta t}(\mathbf{v}_A(\tau) - \mathbf{v}_A^{opt})d\tau\| \\ &\leq \|\mathbf{v}_A(t) - \mathbf{v}_A^{opt}\|\Delta t \\ &\leq \sqrt{\max(\dot{\mathbf{u}}_{A_x}^{max}, \dot{\mathbf{u}}_{A_x}^{Max})^2 + \max(\dot{\mathbf{u}}_{A_y}^{max}, \dot{\mathbf{u}}_{A_y}^{Max})^2}\Delta t \end{aligned}$$

where the first inequality holds if we suppose that the velocity controller drives the current velocity to the desired one in a monotonous way. Then, if we set Δt small enough, this error is not significant (see ***Fig.* 2.9**). In addition, e does not accumulate along time since a new $\mathbf{v}_A^{opt}$ is estimated at each iteration, as we have seen above.

Extension to non-holonomic robots

The algorithm described above is applicable to any robot control model, as long as $\mathcal{U}_A$ and $ACV_A(t)$ can be well specified and computed. The latter will be different for each model, since it represents the result of an integration of valid controls for A. We now focus on *differential drive robots* (DDRs) because they are the most common in this research area. The controls of these robots are defined as follows:

$$\mathbf{u} = \begin{bmatrix} \ell \\ \omega \end{bmatrix},$$

Algorithm 1 Algorithm for choosing the optimal velocity

Input: τ, $\Delta\tau$, τ_{min}, t_c, nO, nmO, $\mathcal{O}$
Output: $\mathbf{v}_A^{opt}$

calculate $\mathbf{v}_A^{pref}$
compute $ACV_A(t)$, $ACV_A^{view}(t_c)$
compute VO_A^{τ}, the set of VOs for the robot induced by $\mathcal{O}$ within time window τ
$\mathcal{LVO} = \{vo \in VO_A^{\tau} \mid vo \cap ACV_A^{view}(t_c) \neq \emptyset\}$
sort $\mathcal{LVO}$ by t_{start}
if ($\mathbf{v}_A^{pref} \in ACV_A(t) \backslash VO_A^{\tau}$) **then**
    return $\mathbf{v}_A^{pref}$
$\mathcal{C} = \emptyset$
for all $lvo \in \mathcal{LVO}$ **do**
    $\mathcal{TC}$ = Candidates generated by lvo inside $ACV_A(t)$
    $\mathcal{C} = \mathcal{C} \cup \mathcal{TC}$
sort $\mathcal{C}$ by distance to $\mathbf{v}_A^{pref}$
return if there exists $c \in \mathcal{C}$, the best viable candidate
repeat
    $\tau = \tau - \Delta\tau$
    update $\mathcal{LVO}$ by removing VOs with t_{start} after τ (relaxing constraints)
    update $\mathcal{C}$, by removing missing candidates and updating their viability
    return if there exists $c \in \mathcal{C}$, the best viable candidate
until ($\forall vo \in \mathcal{LVO} \mid vo_{t_{start}} < \tau_{min}$ **or** $\mathcal{LVO} \neq \emptyset$)
update $\mathcal{LVO}$ by removing VOs generated by the farthest $nO - nmO$ obstacles (relaxing constraints)
update $\mathcal{C}$, by removing missing candidates and updating their viability.
if there exists $c \in \mathcal{C}$ **then**
    return the best viable candidate
else
    return the maximum attainable velocity that approaches $\mathbf{v}_A$ at zero velocity.

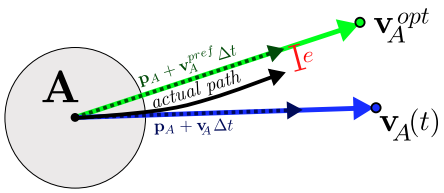


Figure 2.9: In black, the path that A actually traces in the time period Δt by changing its velocity from $\mathbf{v}_A(t)$ to $\mathbf{v}_A^{opt}$. In red, the magnitude of the error e, which is bounded by the magnitude of the distance between $\mathbf{p}_A(t) + \mathbf{v}_A(t)\Delta t$ (dotted blue), and $\mathbf{p}_A(t) + \mathbf{v}_A^{opt}\Delta t$ (dotted green).

with ℓ the linear velocity, and ω the angular velocity. In this case $|\mathbf{u}|$ corresponds to $|\ell|$. As in the holonomic case, we assume that $\dot{\mathbf{u}}$ corresponds to the acceleration taken by A when having to apply the control $\mathbf{u}$, bounded between the maximum acceleration reachable by the robot, $\dot{\mathbf{u}}_A^{Max}$ and its maximum deceleration $\dot{\mathbf{u}}_A^{max}$. In this way, $\mathcal{U}_A$ is defined as:

$$\mathcal{U}_A = \{\mathbf{u} \in \mathbb{R}^2 \mid |\ell| \leq s_A^{max} :: \dot{\mathbf{u}}_{A_\ell}^{max} \leq \dot{\ell} \leq \dot{\mathbf{u}}_{A_\ell}^{Max}; \dot{\mathbf{u}}_{A_\omega}^{max} \leq \dot{\omega} \leq \dot{\mathbf{u}}_{A_\omega}^{Max}\},$$

and then we transform these constraints from the ℓ, ω plane to the cartesian velocities plane, ***see Fig 2.10***.

We get a rough estimation of the set of admissible velocities by integrating $\mathcal{U}_A$ in order to get $ACV_A(t)$. It is important to observe that the geometry is absolutely not the one of $\mathcal{U}_A$, as in the holonomic case. The set $ACV_A(t)$ is still centered on $\mathbf{v}_A(t)$ but its orientation is aligned with the orientation of the robot and not with the Cartesian plane, while the maximal values for the angular velocity define a cone-like shape of possible cartesian points to reach, ***see Fig 2.11***.

The next chapter will show how this deterministic approach was implemented and will describe the results we obtained in simulation.

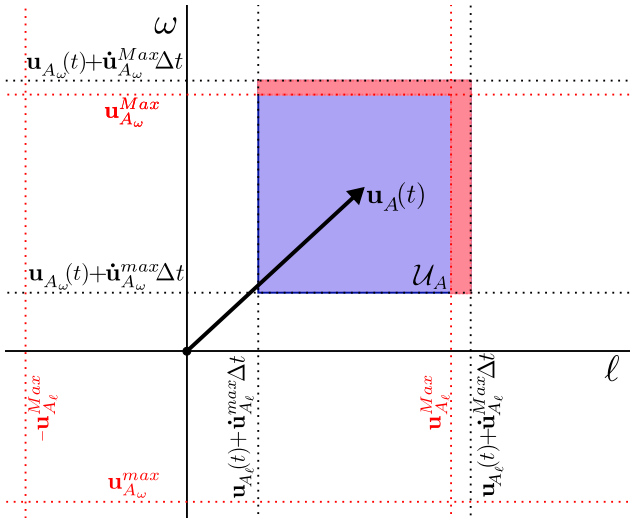


Figure 2.10: The set of controls $\mathcal{U}_A$ bounded by the integration of its maximum accelerations and decelerations in both ℓ and ω, shown as dotted black lines. Also, we depict the maximum and minimum velocities in ω, as horizontal dotted red lines, and the maximum velocity norm of ℓ restrict $\mathcal{U}_A$. The latter corresponds to s_A^{max} and is represented as the bounded area between the two vertical dotted red lines.

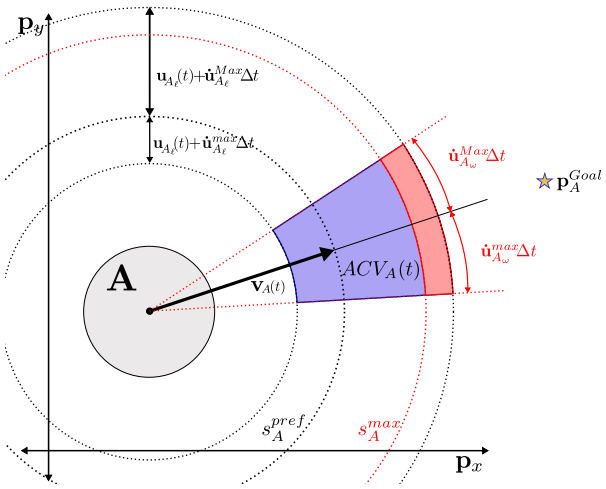


Figure 2.11: Integration of $\mathcal{U}_A$ in order to get $ACV_A(t)$ in the Cartesian plane, aligned with the robot orientation. In straight dotted red lines, the constraints induced by the maximum acceleration and deceleration in ω. ℓ is bounded by s_A^{pref}, shown as the dotted red circle, as well as by the maximum acceleration and deceleration of ℓ, shown as dotted black circles of smaller and larger radius.

Chapter 3

Implementation and results of Deterministic Approach

3.1 Data inputs and velocity obstacle representations

To describe the most probable trajectories that a mobile obstacle will trace, a set of $2D$ points corresponding to a trajectory of duration T is given for each model. The following format of plain text data is used.

```
ID
N_MODELS T
x y vx vy

... ... ... ...

x y vx vy
# PROB

...

x y vx vy
... ... ... ...
x y vx vy
# PROB
.
```

For each observation (every time data is received), the format first describes, on a single line, the *ID* of the mobile obstacle. Then, in the next line, it gives the number of predicted models and the duration τ, shown as *T*, during which they will be traced. For each model, a sequence of points is given, one per line, described by its coordinates x, y

and the variances var_x, var_y on this position. To complete the description of a model, the following line has a *wildcard* character $\#$, and the probability of occurrence *PROB* of the model. Finally, the character **.** completes the input of that observation.

We have implemented a C++ library that we called *Simulator*, with which we can simulate/represent a dynamic environment and control one or multiple robots, choosing at each time step a velocity that will bring them closer to their $goals$, while avoiding collisions between them, and with other mobile or static obstacles. This class receives the data (about obstacles) either by reading a file or by streaming, and *packs* it in a structure that assigns to each obstacle a time horizon τ, and communicates these data to each robot. This structure stores a set of predicted trajectory models, each corresponding to a probability of occurrence and it is described as a sequence of points x, y, with their variances var_x, var_y. In the same way, *Simulator* can receive, as a data input, a set of structures as described above, generated by an external program. With this class, we can simulate a centralized observation system, i.e., a system responsible for receiving and processing the input data that represent the positions of all the obstacles and their prediction models. It is also responsible for giving the notion of fixed obstacles, and for updating the status of the robots, i.e., it communicates to each robot the position of its nO nearest obstacles (mobile and fixed), and indicates it when it is ready to update its velocity.

Discretization of trajectories

For each trajectory model, we can model up to $p-1$ chords, where $p \geq 2$ is the number of points that describe the trajectory by a piecewise linear model. In the deterministic modelling, we use only one model, the one with the highest probability, and we do not take into account the variance of the points.

Because we know the time span τ corresponding to the prediction, it is possible to estimate the velocities of each sub-trajectory. For practical purposes, we assume that the speed is constant throughout the prediction and that the points are evenly distributed. Therefore we can easily estimate a set of up to $p-1$ chords, which will be the basis for constructing the $QVOC$. In ***Fig. 2.3***, we can see the way in which a trajectory is discretized, given as input data a set of points along that trajectory. Each chord is indexed by i and will be described by the following attributes:

τ_i: the time at which the obstacle is predicted to begin tracing the chord i.

t_{dur}: the duration of the tracing. For simplicity, we assume that all the chords will be traced with the same duration. It corresponds to $\tau_{i+1} - \tau_i$.

$velocity$: the predicted velocity with which the obstacle will trace the chord.

p_{start}: starting point of the chord, which corresponds to the point at which the obstacle is predicted at τ_i.

Modelling of QVOs

As we proposed in the previous chapter, we use $QVOs$ to represent the VOs that describe each predicted chord between times τ_i, τ_{i+1}, for each obstacle. For practicity in the construction, we do not link the $QVOs$ of the same chain in the same structure, but we store them as independent objects.

Based on observations made on the human behavior, we define a *comfort radius* for the robot. With this radius, the robot avoids passing very close to the obstacles, since otherwise, due to the construction of the VOs, the robot avoids collisions by passing tangentially to the obstacles. This is because $\mathbf{v}_A^{opt}$ is chosen on $\mathcal{B}ACV$.

Each QVO consists of the following attributes:

τ_i: the *start time* that bounds the QVO, the same as τ_i of the chord.

τ_{i+1}: the *end time*, that corresponds to τ_{i+1} of the chord that it describes.

$points$: array of four $2D$ points that are the vertices bounding the QVO. Joining them in the order they are stored, and joining the endpoints, we get a quadrilateral.

The construction of the QVO is then very simple. We only have to calculate the four points that bound the VO, that is, two points corresponding to the segment $\mathbf{p}_A + \mathbf{v}_B + S(\frac{\mathbf{p}_B - \mathbf{p}_A}{\tau_i^-}, \frac{cr_{A,B}}{\tau_i^-})$, and two more points corresponding to the segment $\mathbf{p}_A + \mathbf{v}_B + S(\frac{\mathbf{p}_B - \mathbf{p}_A}{\tau_{i+1}^+}, \frac{cr_{A,B}}{\tau_{i+1}^+})$, ***see Fig. 2.2***. Finally, we store these $QVOs$ in a vector, VO_A^τ, as described in line 3 of ***Algorithm 1***. Then we compute $\mathcal{LVO}$ and sort it in ascending order by τ_i, as seen in lines 4 and 5. For our experiments we consider $t_c = \infty$, that is, we omit the construction of $ACV_A^{view}(t_c)$, and use all the generated $QVOs$.

Virtual Obstacles

To model the *VOC* corresponding to the discretized predicted trajectory of an agent B, first a *virtual* mobile agent $\mathbf{B}_v(\tau_i)$ is created for each chord, starting from an initial point $\mathbf{p}_B(\tau_i)$, such that moving at a constant velocity $\mathbf{v}_B(\tau_i)$, it traces the chord between the times $[\tau_i, \tau_{i+1}]$. This is similar to the *Equivalent Linear Velocity Obstacle*, $ELVO(B, \tau_i)$ introduced in [15], but replacing the tangent to the trajectory with the chord between two points of it; with an infinite number of tangents and chords, both cases would be equivalent, ***see Fig. 3.1.a***.

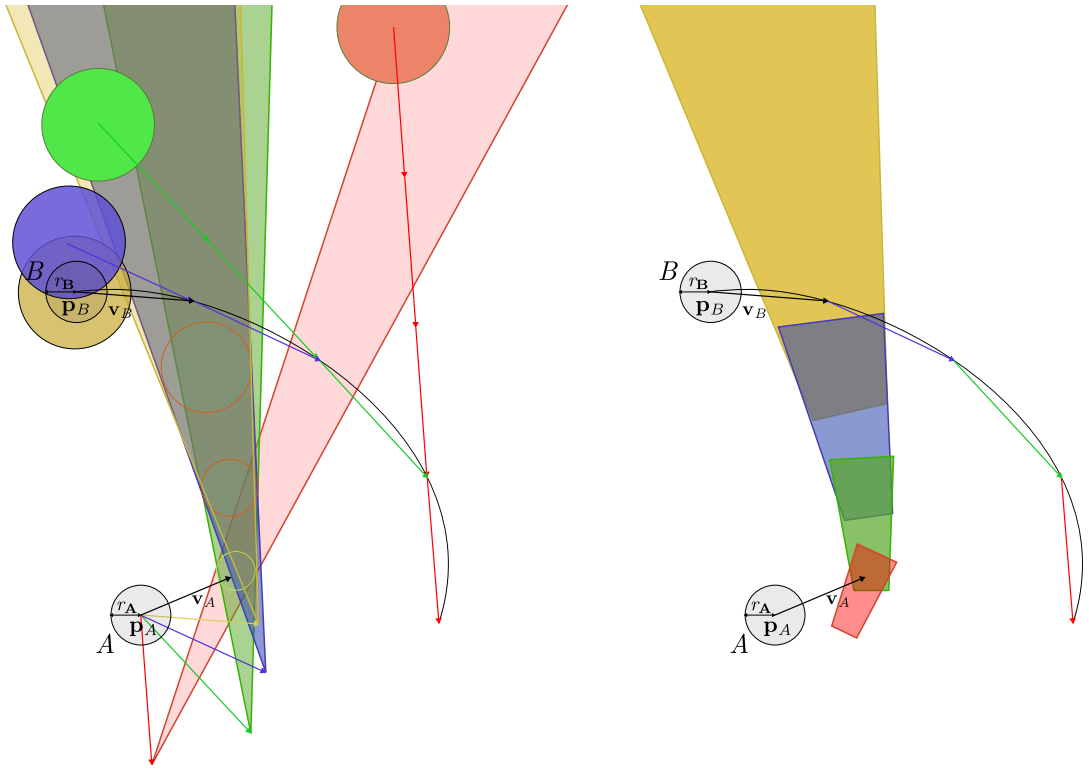


Figure 3.1: **a)** Construction of $VOC_{A|B}$. Objects in red, green, blue and gold correspond to *virtual agents*[15], their trajectories and velocities. **b)** $VOC^{\tau}_{A|B}$ within a time window τ, modelled with quadrilaterals, $QVOC^{\tau}_{A|B}$.

Once $ELVOs$ are estimated for each chord of the prediction, the VOC can be constructed by truncating the shape of each according to the t_{dur} corresponding to the tracing of the chord. ***Fig. 3.1.b*** shows the *Quadrilateral VO Chain* of A induced by B, for a time horizon τ, $QVOC^{\tau}_{A|B}$.

3.2 Evaluation of the feasibility of the candidates

If $\mathbf{v}_A^{pref}$ is in conflict, then we must evaluate the feasibility of the candidates to choose the best of them, as shown in lines 13, 18 and 23 of the ***Algorithm 1***.

To avoid oscillations when choosing $\mathbf{v_A^{opt}}$ [9], we weight the distance between a candidate $\mathbf{c}$ to the preferred velocity $\mathbf{v}_A^{pref}$ with the distance between $\mathbf{c}$ to the current velocity $\mathbf{v}_A$. To achieve more continuous velocities, i.e., less oscillations, we use an adjustable parameter α, to give more or less priority to velocities close to $\mathbf{v}_A$. Hence, we redefine the distance function for a candidate $\mathbf{c}$ to $\mathbf{v}_A^{pref}$ as follows:

$$wDist(\mathbf{c}, \mathbf{v}_A^{pref}, \mathbf{v}_A, \alpha) = \alpha \| \mathbf{c} - \mathbf{v}_A^{pref} \| + (1-\alpha) \| \mathbf{c} - \mathbf{v}_A \|,$$

where $\alpha \in [0, 1]$. In our implementation we set $\alpha = 0.8$. The optimization problem becomes:

$$\begin{aligned} \mathbf{v}_A^{opt} &= \underset{\mathbf{v} \in ACV_A(t) \setminus VO_A^{\tau}}{\operatorname{argmin}} \alpha \| \mathbf{v} - \mathbf{v}_A^{pref} \| + (1-\alpha) \| \mathbf{v} - \mathbf{v}_A \| \\ &= \underset{\mathbf{v} \in ACV_A(t) \setminus VO_A^{\tau}}{\operatorname{argmin}} wDist(\mathbf{v}, \mathbf{v}_A^{pref}, \mathbf{v}_A, \alpha). \end{aligned}$$

The candidates are stored in a *double ended queue* **(deque)**, for the purpose of accessing them in amortized constant time, *O(1)*. We denote this *deque* as $\mathcal{C}$, as in ***Algorithm 1*** from line 8, and sort each of its elements c by the value of the *weighted distance* function evaluated on them. The sorting of the structure is carried out with a computational complexity of $N * \log(N)$. In case a candidate is generated as the intersection of two $QVOs$, we will call them *parent_1* and *parent_2*. Each candidate is described by the following attributes:

vx: Coordinate x of the candidate in the velocity space.

vy: Coordinate y of the candidate in the velocity space.

$distance$: Value of the *weighted distance* function of the candidate: $wDist(c, \mathbf{v}_A^{pref}, \mathbf{v}_A, \alpha)$.

$belongs$: List of indices in $\mathcal{LVO}$, which identify the $QVOs$ that make the candidate not viable, i.e., the $QVOs$ in $\mathcal{LVO}$ that contain c.

$parent\_1$: Index that identifies one of the two $QVOs$ that generate the candidate by an intersection of both, if it is the case. Otherwise it is the index of the QVO that intersects with $\mathcal{B}ACV$, or the QVO on which it projects orthogonally.

$parent\_2$: Index of the other parent that generates the candidate by intersection, if applicable.

To know if a candidate is viable, we use a version of the *Crossing Test* algorithm [19], which consists in evaluating whether a point is inside a polygon or not, by tracing a line from the point, in some direction towards infinity. If the candidate is within the polygon, the number of crosses of this line with the segments of the polygon will be **odd**, otherwise it will be **even**.

In our implementation, the algorithm consists of two phases. The first step is to validate whether the candidate c is on any of the four segments that delimit the QVO. If so, we say that the point is outside the quadrilateral. The implementation of this part of the algorithm consists in evaluating if any of the segments between two points $\mathbf{a}$ and $\mathbf{b}$ delimiting the quadrilateral contains c, i.e., $abs((c_x - \mathbf{a}_x)(\mathbf{b}_y - \mathbf{a}_y) - (c_y - \mathbf{a}_y)(\mathbf{b}_x - \mathbf{a}_x)) < \epsilon$.

Due to the possible numerical errors, we allow a tolerance of $\pm\epsilon$ to the result value of $(\mathcal{c}_x - \mathbf{a}_x)(\mathbf{b}_y - \mathbf{a}_y) - (\mathcal{c}_y - \mathbf{a}_y)(\mathbf{b}_x - \mathbf{a}_x)$, with $\epsilon \approx 10^{-6}$.

The second phase of our implementation consists in counting the number of crosses made by a ray to infinity in the direction v_x^+, $\overrightarrow{r}$, that starts in $\mathcal{c}$, on each side $\overline{\mathbf{ab}}$ of the QVO, consisting of a pair of consecutive points $\mathbf{a}, \mathbf{b}$. Because the coordinate $\overrightarrow{r}_y$ is constant along the ray, we first evaluate that $\mathcal{c}_y \in [\mathbf{a}_y, \mathbf{b}_y]$. Otherwise, there are 0 crosses between $\overrightarrow{r}$ and $\overline{\mathbf{ab}}$. Then, if the coordinate v_x of the point of intersection $\mathbf{p}_{int}$ between $\mathbf{r}$ and $\overline{\mathbf{ab}}$, i.e., $\mathbf{p}_{int_x} = \mathbf{a}_x + \frac{(\mathcal{c}_y - \mathbf{a}_y)(\mathbf{b}_x - \mathbf{a}_x)}{(\mathbf{b}_y - \mathbf{a}_y)}$, is greater than or equal to $\mathcal{c}_x$, there will be an intersection.

In this process, we take advantage of the *Crossing Test* that is made on each candidate, with each $QVO \in \mathcal{LVO}$, in order to fill what we refer to as its $belongs$ list. This is done with the purpose of knowing exactly which of the $QVOs$ contains the candidate, making it not viable. Then, if we relax the constraints, it is easier to update its viability.

Relaxation of constraints

Once we have checked all the candidates, and if none has been found viable, we proceed to relax the constraints. That is, to remove some of the $QVOs$ of $\mathcal{LVO}$, in order to increase the *feasible area of attainable velocities*, i.e., $ACV_A(t) \backslash VO_A^\tau$. We first relax constraints with respect to time, lines $14 - 19$ of ***Algorithm 1***. If it is not enough, we reduce the number of visible obstacles, lines $20 - 23$ of ***Algorithm 1***.

Both ways of relaxing the constraints result in the reduction of $\mathcal{LVO}$. By having less $QVOs$, some of the candidates will be eliminated, as one or both of their *parents* will disappear. In the same way, some of the candidates will *survive* and may have their status changed to viable, so they could become eligible as $\mathbf{v}_A^{opt}$.

Constraints on time horizon

As a first intent, we eliminate the $QVOs$ that represent longer *time to collision*, i.e., those such that their τ_i is greater than a given τ. Remember that $\mathcal{LVO}$ is sorted in ascending order by τ_i, in order to quickly find the candidates that meet the condition in each step of relaxation in the time horizon. ***See Algorithm 1***, lines $14 - 19$. The comparison can be done with linear computational complexity O(N), with N as the number of candidates, by checking one by one the elements in $\mathcal{LVO}$, or with logarithmic complexity, O(log N), by doing a binary search. Remember that in each relaxation, we must update $\mathcal{C}$ and the viability of the candidates.

Constraints on visible obstacles

In the case we have not found $\mathbf{v}_A^{opt}$ until now, our second intent is to reduce the number of visible obstacles to eliminate more elements of $\mathcal{LVO}$, ***see Algorithm 1*** lines 20 - 23.

Initially, the robot *can see* up to nO obstacles, and after reducing the number of these, the robot *can see* up to nmO obstacles. These should be the nearest nmO obstacles. In order to select them, we use the library *boost for geometry* [20]. With this library, we implemented an **R-tree**[21] on the set of points that represent the obstacles, i.e., their centroids. An **R-tree** allows us to make queries with complexity of O(log N). It is an optimized data structure for query of *spatial data*. This allows us to efficiently consult the nearest nmO visible obstacles.

3.3 Experiments with *Simulator*

As mentioned before, the *Simulator* class provides what is necessary to design and simulate an environment of multiple mobile agents (robots or humans), and fixed obstacles. In order to visualize the experiments, we have developed an interactive application with *OpenFrameworks development toolkit*, which allows us to work on different platforms, because it is *cross-compatible*. It currently supports five operating systems (Windows, OSX, Linux, iOS, Android) and five IDEs (XCode, Code::Blocks, and Visual Studio, Eclipse and QTCreator) [22]. We use the latter to develop the application.

The simulator consists of a basic visualization of the elements of the environment. Remember that both robots and humans are represented in the Simulator class as disk-shaped objects, as shown in ***Fig. 3.2*** and ***Fig. 3.3***.

Fig. 3.2 shows a simulated experiment of a robot A, in white and purple. At the origin of the cartesian frame, shown as a cross at the center of the figure, a blue circle represents the zone corresponding to $\mathbf{p}_A^{Goal}$, with arrival tolerance (within the disk).

Fig. 3.3 shows some of the attributes of the robot. The red and white concentric squares correspond to the set of velocities (vx, vy) with magnitude $s^{pref}_{A_{vx,vy}}$ and $s^{max}_{A_{vx,vy}}$, respectively. The lines that start from the center of the robot, in red and blue, correspond to the current velocity $\mathbf{v}_A$ and the preferred velocity $\mathbf{v}_A^{pref}$, respectively. In light yellow, and centered on $\mathbf{v}_A$, is $ACV_A(t)$. Note that it is truncated with the maximal velocity $s^{max}_{A_{vx,vy}}$.

The class *Simulator* allows us to execute experiments without having to visualize them. This is very useful to measure the performance of the ***Algorithm 1***.

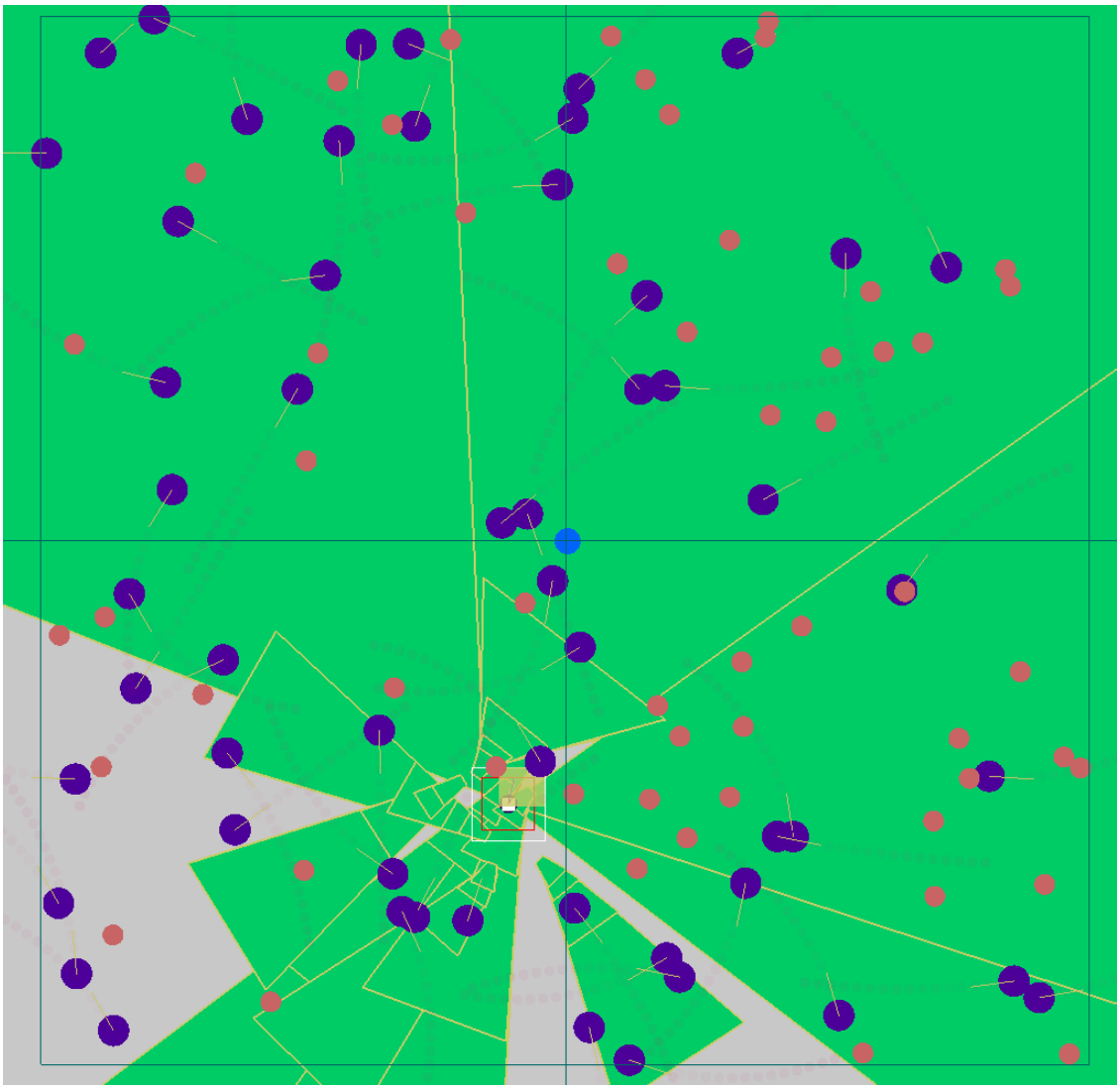


Figure 3.2: Experiment with a holonomic robot shown in white and purple, with 55 simulated non-controlled mobile obstacles (humans) in purple and their exact predicted positions within a time horizon of 5 seconds, shown in magenta gradient. In coral, we depict 55 fixed obstacles. Gray areas (in the velocity space) are unconstrained, while green areas are not, corresponding to the VOs induced by the obstacles.

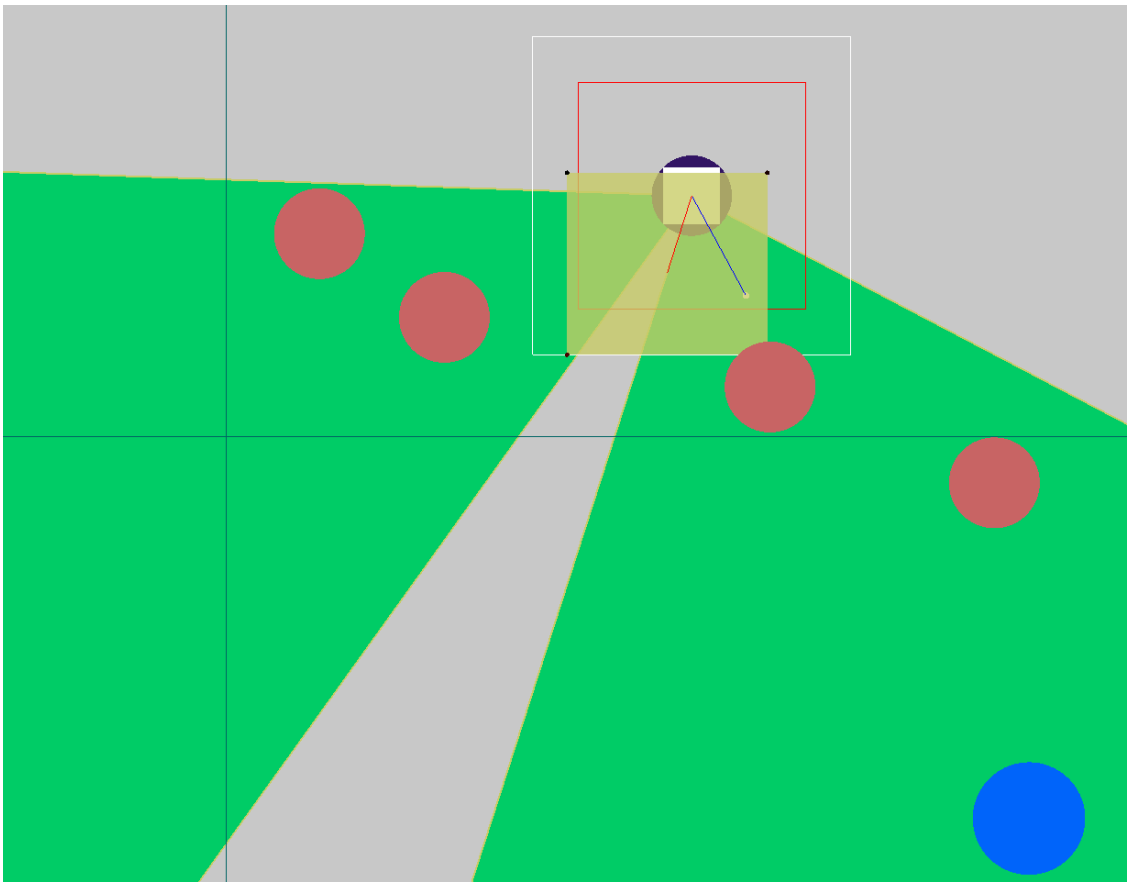


Figure 3.3: Zoom on the robot, shown as a purple and white circle. Lines in red and blue correspond to the current velocity $\mathbf{v}_A$ and the preferred velocity $\mathbf{v}_A^{pref}$. In order to avoid collisions, the robot takes velocities on the border of VO_A^τ, shown as green areas.

Setup of the experiments

In order to measure the performance of the algorithm, a *base case* was defined. This consists of a series of parameters of configuration of the algorithm, such that they make a robot A reach its *Goal* in a considerable average time, and with a low number of iterations in *conflict* (collision), i.e., less than 1%.

All the experiments were executed on a machine with 24 CPUs at $2600\ MHz$, with $32910284KB$ of RAM, running a 64-bit Linux operating system and having $93180\ KB$ of Swap space. The experiments were performed in a simulated environment consisting of a space of squared shape and dimension $400m^2$, where a set of randomly arranged humans moves, (*see Appendix A*). To keep the density of the environment constant, humans are restricted to moving only within it. The initial position of the robot is configured by randomly choosing a polar angle in $[-\pi, \pi]$. Then the robot is placed on the circle inscribed to

the environment, in the direction of the chosen angle, and looking towards the center of it. $\mathbf{p}_A^{Goal}$ is also on the circle, in a position diametrically opposite to the initial position of A.

Now we describe the parameters that were used to simulate the robot and the humans. In order to adjust these parameters as realistically as possible, we take as a basis the physical configuration of the *Turtlebot 2*[23], resulting in the following parameters:

$Radius$: $0.177m$.

Max ℓ: $0.7m/s$.

Max ω: $\pi rad/s$.

Max $\dot{\ell}$: $1.26m/s^2$.

Max $\dot{\omega}$: $5.65rad/s^2$.

We assume that the maximum decelerations are symmetrical to the maximum accelerations, i.e., these have the same magnitude but in the opposite direction. We also assume that they correspond to 1.8 times the maximum velocities. Because *Turtlebot 2* is a *DDR* robot, we use some of its specifications for the purpose of modelling a *holonomic* robot, resulting in parameters for this case as follows:

$Radius$: $0.177m$.

Max v_x: $0.7m/s$.

Max v_y: $0.7m/s$.

Max $\dot{v}_x$: $1.26m/s^2$.

Max $\dot{v}_y$: $1.26m/s^2$.

The following list shows the configuration parameters that were used for the experiments and their *default* value (base case):

nH: the number of humans in the environment, *(150 humans)*.

$nOPercentage$: percentage of obstacles that the robot *can see* with respect to nH, $(nO = nH * (nOPercentage/100))$, (50%).

$ComfortRadScale$: the scale of the comfort radius that is used with respect to the original radius of the robot, (1.1).

$RestAccScale$: the scale of the restrictions in acceleration, with respect to the maximum velocity, (1.8).

$Segments$: number of chords into which a predicted trajectory is divided, $(5\ segments)$.

τ: duration of predictions, $(5.0\ seconds)$.

Δt: the duration of the time increment for the simulation, $(0.35\ seconds)$.

In this implementation, the number of obstacles that the robot *can see* (nO), directly affects the minimum number of obstacles that the robot *must see* (nmO) when the constraints are relaxed. For these experiments, we set $nmO = nO * 0.2$, so as not to relax excessively the constraints. It is also necessary to mention that the scale with respect to the restrictions in acceleration influences both velocities, i.e., we use the same scale for v_x, v_y in the holonomic case, and the same scale for ℓ, ω for *DDR* robots.

In order to compare the performance of the robot with different setups, the experiments were performed by varying each of the 7 parameters mentioned above, in at least five different configurations, the *base case* being one of them. We use three metrics to measure the overall performance of the algorithm: *Percentage of iterations in conflict (collision)*; *Average time per iteration* (measured in milliseconds); and *Number of iterations made until reaching the Goal*. Below we show a summary of the results obtained by varying the configuration parameters of the robot for the *holonomic* case.

Experiments with the holonomic robot

8 thousand executions of the algorithm were performed with the base setup. ***Fig. 3.4*** shows some *boxplot* graphs to interpret the three proposed metrics. A *boxplot* is a useful statistical graph to visualize the distribution of a data set. The upper and lower lines that delimit the *box* represent the positions of the $3rd$ and $1st$ quartiles, Q_3, and Q_1 respectively. The red line shows the mean of the data. The points represented as crosses and circles are *outliers* data. We define *IQR* as the *interquartile range*, and it refers to the distance between Q_1 and Q_3. Then, the *outliers* are beyond $1.5 * IQR$, either below Q_1 or above Q_3. In the same way, the *extremely outlier* data (circles) are located beyond $3 * IQR$ below Q_1or above Q_3.

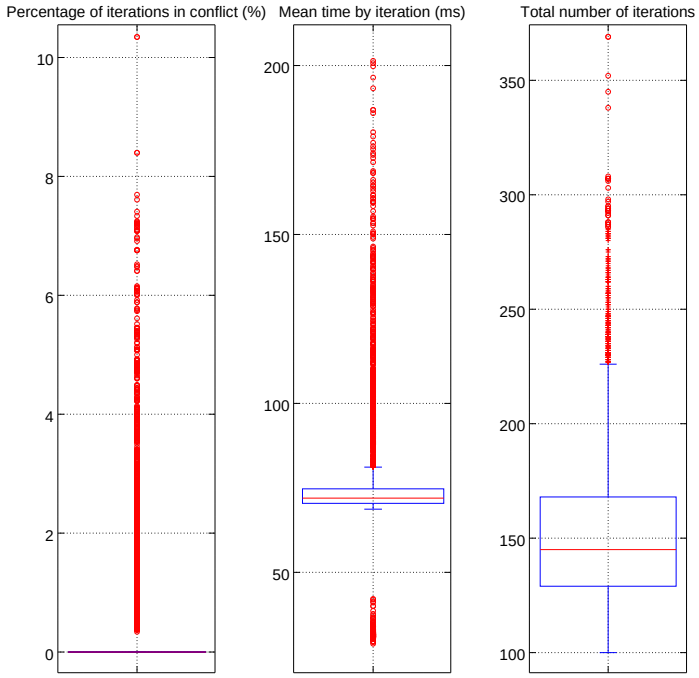


Figure 3.4: Three *boxplots* representing the distribution of the performance metrics proposed on the base case.

Fig. 3.4 is useful as a reference for the interpretation of subsequent *boxplots*. In the case of ***Fig. 3.4.left***, the *box* is completely *flat* because the vast majority of results, i.e., at least 75% of them, result with 0% of iterations in conflict. The *outliers*, displayed in red, show that in the base case, the maximum percentage of iterations in collision is 10.34%. Its mean is 0.37%.

In ***Fig. 3.4.center***, the box is partially *flattened*. Quartiles Q_1, Q_2 (median), Q_3, have values of $70.42, 71.96, 74.71$, respectively. It has a large number of outliers, both below Q_1, and above Q_3, while the mean has a value of $70.91\ (ms)$.

Finally, in ***Fig. 3.4.right***, quartiles Q_1, Q_2 (median), Q_3, have values of $129, 145, 168$, respectively. All its outliers are above Q_3. Its mean has a value of 152.82. Based on these results, we can compare the performance of the experiments by varying each of the 7 parameters. For these experiments, each setup was executed 1000 times, in order to obtain sufficiently dense data sets.

Variation of the parameter nH

The number of simulated humans directly influences the density of the environment. Since its dimensions are constant $(20m * 20m)$, the density corresponds to $\frac{nH}{400}\frac{humans}{m^2}$. Recalling that the base case has a total of $150\ humans$, then its density is $0.375\frac{humans}{m^2}$. The values in which the parameter was varied nH and their respective densities are the following: 50: $0.125\frac{humans}{m^2}$; 75: $0.1875\frac{humans}{m^2}$; 100: $0.25\frac{humans}{m^2}$; 150: $0.375\frac{humans}{m^2}$ (base case); 200: $0.5\frac{humans}{m^2}$.

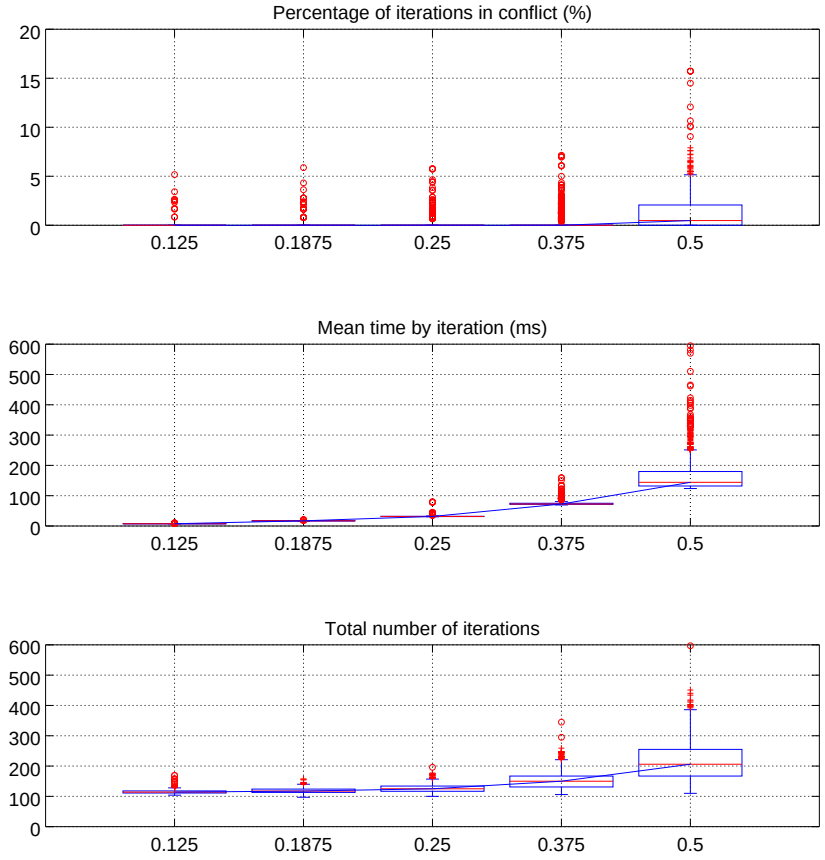


Figure 3.5: Performance metrics on the testing sets by varying the parameter nH. The x axis shows the value of the human density of the environment, measured in $\frac{humans}{m^2}$. With $0.375\frac{humans}{m^2}$ in the base case.

In ***Fig. 3.5***, we can see the impact of the number of humans on performance metrics. ***Fig. 3.5 top*** shows that the percentage of iterations in conflict is negatively affected, as ex-

pected, by increasing the density of the environment. The first four density values show the majority of their records with 0% of iterations with collision, with maximum *outliers* between 5% and 7.5%. On the other hand, having a density of $0.5\frac{humans}{m^2}$, the median has the value of $Q_2 = 0.48$, and $Q_3 = 2.07$. Although the percentage of iterations with collision is small, since it is the most important metric, it must always be minimized.

The average time per iteration is also affected by increasing the density, because there is a greater probability that the *feasible area of attainable velocities* be totally obstructed. That means that the algorithm would try to relax in greater proportion the constraints to find $\mathbf{v}_A^{opt}$, causing an extra cost in the computing time. In ***Fig. 3.5 center***, we can see that the average time per iteration is acceptable, i.e., in all configurations, Q_3 stays below $\Delta t = 350\ ms$, taking the values $7.558\ ms, 17.35\ ms, 31.98\ ms, 75.04\ ms, 179.74\ ms$, respectively. However, the maximum outlier observed for $nH = 200$ reaches the value of $594.91\ ms$, also showing a significant number of them above Δt. The other configurations show maximum outliers smaller than $160\ ms$, therefore, they remain within the execution parameters for the base case.

The number of iterations made until reaching the *Goal* also increases as the density of humans in the environment increases. In ***Fig. 3.5 bottom***, we can see that in experiments with low density, the number of iterations to reach the goal is relatively small, with outliers smaller than 200. For the base case, as for 200 humans, the number of iterations increases on a large scale, reaching outliers of up to 345 y 597, respectively.

Variation of the parameter nOPercentage

The maximum percentage of simulated humans that the robot *can see*, is used not to saturate the *feasible area of attainable velocities*, so that it is computationally less expensive to calculate $\mathbf{v}_A^{opt}$. ***Fig. 3.6*** shows the behavior of the three performance metrics by varying the parameter $nOPercentage$. Recall that the base case takes into account 150 humans, of which nO corresponds to 50%, and nmO to 10%. Also recall that $nmO = nO * 0.2$. The values in which the parameter $nOPercentage$ was varied, and their respective nO and nmO, are the following ones: 10%: 15 humans, 3 humans; 25%: 37 humans, 7 humans; 50%: 75 humans, 15 humans (base case); 80%: 120 humans, 24 humans; 100%: 150 humans, 30 humans.

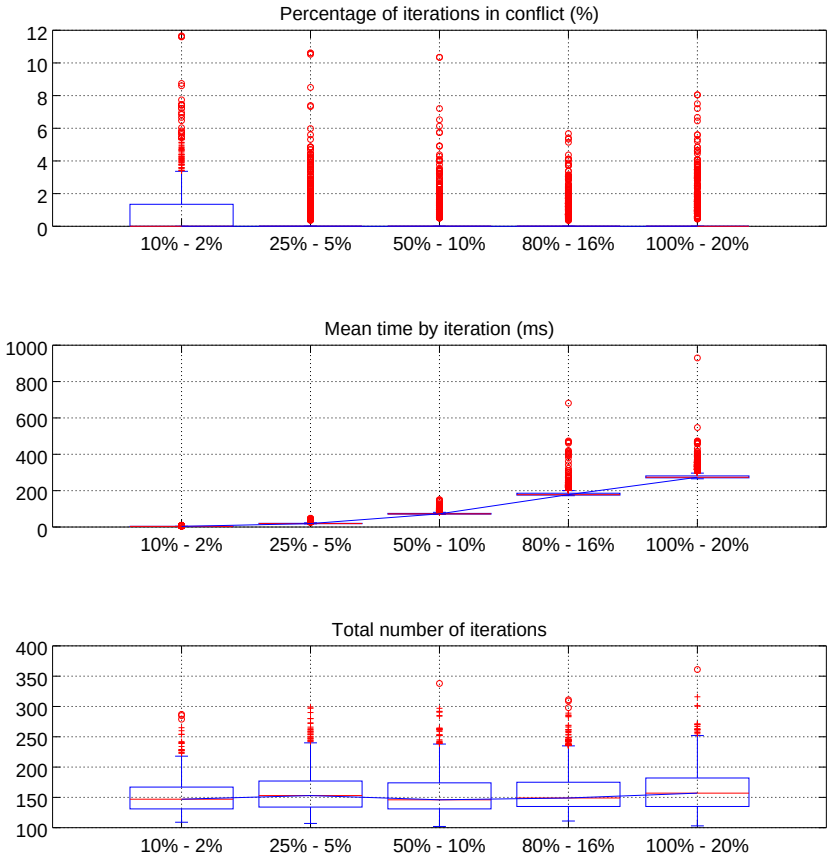


Figure 3.6: Performance metrics varying the parameter $nOPercentage$. The x axis of the three graphs shows the percentages that represent nO and nmO over the number of humans, (150). With $50\% - 10\%$ in the base case.

Fig. 3.6 top shows that, in general, the percentage of iterations in conflict is mostly 0% when the robot *can see* at least 25% of the humans. With 10% of the humans, the robot has an average of 0.86% iterations in conflict, while for $25\%, 50\%, 80\%, 100\%$ the mean is less than 0.5%. In general, by increasing the percentage of humans that the robot can see, we reduce the percentage of iterations in conflict. When the robot *sees* all the humans, there is a considerable increase in the percentage of iterations in conflict, with respect to the case in which the robot *sees* only 80% of humans. This may be because, as mentioned above, $ACV_A \backslash VO_A^\tau$ can be mostly obstructed, giving the same priority to all humans regardless of their distance and time to collision. At first, it may seem that this is good, but in doing so, humans with lower risk of collision influence the decision making, causing that sometimes $\mathbf{v}_A^{opt}$ is not the best option and that it directs the robot to a *stagnation* impossible to solve. Recall that humans are *blind* and do not interact with the robot or other humans.

In ***Fig. 3.6 center***, we observe that, when increasing the percentage of humans that the robot can see, the average time of computation per iteration increases, as expected, because a greater number of humans will produce a greater number of candidates, and in the same way, $ACV_A \backslash VO_A^\tau$ can be mostly occluded, producing more relaxation steps. In general, in the base case, the average time per iteration is *acceptable* (less than Δt), and in the worst case, when all humans are taken into account, the mean has a value of $285.5\ ms$. However, only the maximum outliers of the first three configurations are less than Δt, with values of $681.44\ ms$ and $929.93\ ms$ for the cases with 80% and 100% respectively.

Finally, in ***Fig. 3.6 bottom***, we can easily see that the number of iterations to reach the goal is practically unaffected by varying the number of visible humans. The means fluctuate between 150 and 162 iterations, while the magnitude of the outliers is affected, although there is no clear pattern on them.

Variation of the parameter $\mathbf{ComfortRadScale}$

The scale of the comfort radius, with respect to the radius of the robot, is a factor that allows the robot to navigate with some slackness among the obstacles. Intuitively, if the scale is 1.0, the comfort radius is equal to r_A, and the robot will do the calculations to avoid the obstacles in a way tangent to its volume. In real life, this is not the case, because people try not to touch each other when they avoid each other. ***Fig. 3.7*** shows the behavior of the three performance metrics by altering the parameter $ComfortRadScale$. Recall that, in the base case, we have used a scale of 1.1, with respect to r_A. The values in which the parameter was varied are the following: 1.0; 1.1 (base case); 1.5; 1.8; 2.0.

In ***Fig. 3.7 top***, we observe that the median of the percentage of iterations in conflict, for the first three configurations of the comfort radius, shows a value of 0%, while for scales 1.8 and 2.0, we get the values of 0.45 and 1.15 respectively. Intuitively, by increasing the comfort radius of the robot, its space of non-conflicting velocities is more restricted, since VOs of greater dimension are generated. As an analogy, if we had a robot represented as a point in space, the VO would only take into account the dimensions of the humans. For practical purposes, the best option is to use a value of 1.1 on the comfort radio scale, according to the experiments. In addition, as explained above, people have greater confidence in walking if they maintain a distance between them and the obstacles.

It is clear that the wider the comfort radius of the robot, the greater the average time per iteration, as it can be seen in ***Fig. 3.7 center***. This is due to what has been commented previously: the velocity space is more occluded by the VOs, then the robot has to relax its constraints to a greater extent.

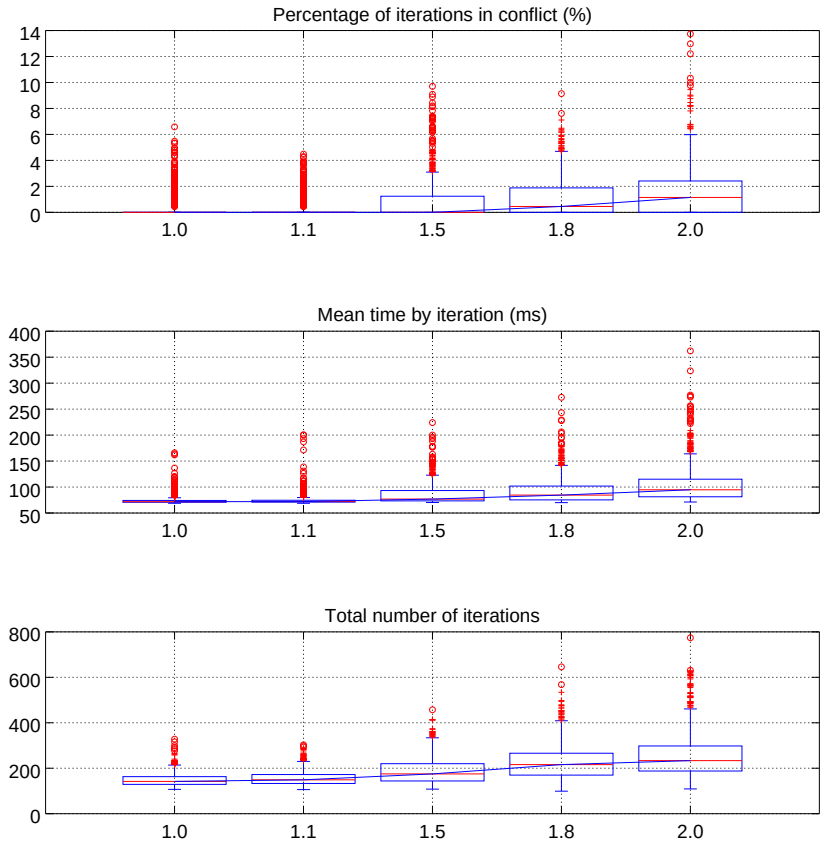


Figure 3.7: Performance metrics by varying the parameter $ComfortRadScale$. The x axis shows the scale of the comfort radius with respect to the radius of the robot. With 1.1 in the base case.

The size of the comfort radius also influences the total number of iterations to reach the *Goal*, as seen in ***Fig. 3.7 bottom***. This is because, in each iteration, the robot is more *cautious* and avoids traversing narrow spaces between obstacles, which causes the robot to try to surround a set of humans, instead of going through it.

Variation of the parameter RestAccScale

The scale of maximum accelerations for the robot, with respect to its maximum control velocities, is a factor that allows the robot to increase its ACV. Intuitively, if the ACV is large, the robot will have a better chance of avoiding obstacles, although its control may

become less continuous. ***Fig. 3.8*** shows the behavior of the three performance metrics by altering the parameter $RestAccScale$. Recall that, in the base case, we take into account a scale of 1.8, with respect to the maximum velocities in the robot controls. The values in which the parameter $RestAccScale$ was varied are the following: 1.0; 1.2; 1.5; 1.8 (base case); 2.0.

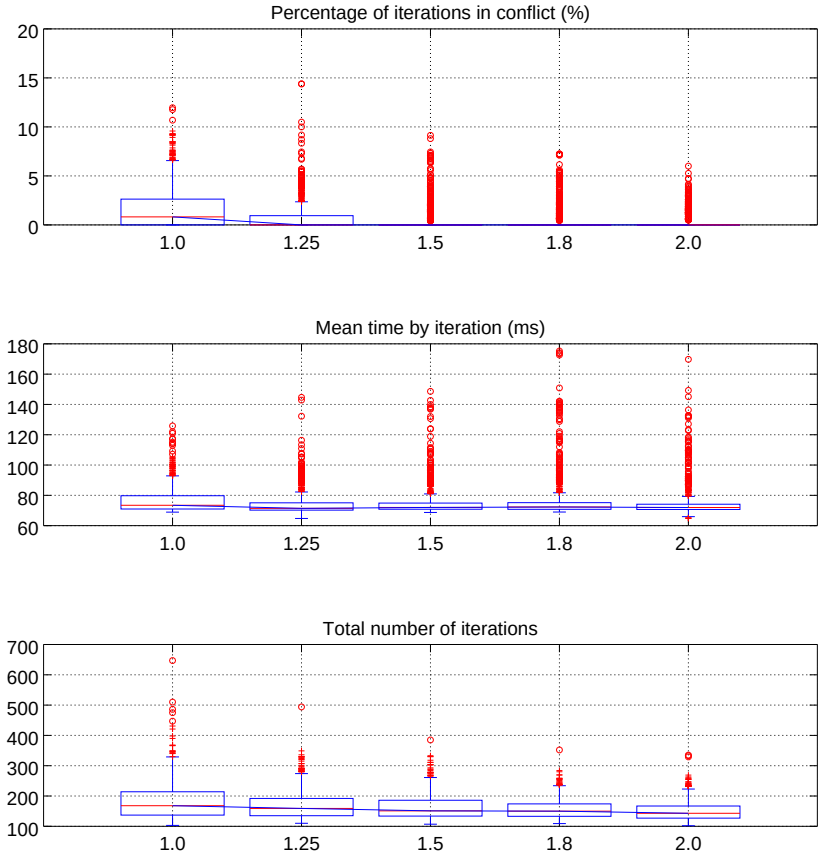


Figure 3.8: Performance metrics by varying the parameter $RestAccScale$. The x axis shows the scale of maximum accelerations for the robot with respect to its maximum control velocities. With 1.8 in the base case.

It is logical to think that the greater the acceleration capacity in the robot controls, the lower the percentage of iterations in collision. In ***Fig. 3.8 top*** it can be seen that, in general, the median percentage of iterations in collision decreases as this parameter grows, because the robot is able to adjust its velocity more freely.

Both the average time per iteration, and the total number of iterations, are not very affected, ***see Fig. 3.8 center, bottom***. The trend is clearer if we look at the outliers: the higher the acceleration capability, the larger the ACV, then the number of candidates is higher, and searching among them for $\mathbf{v}_A^{opt}$ results in a heavier task. We can also observe, based on the outliers, that the total number of iterations tends to be reduced because, as the robot has more options in choosing $\mathbf{v}_A^{opt}$, it can access to a greater extent at speeds with magnitude equal to or greater than the preferred speed of its controls.

Variation of the parameter Segments

The number of segments (chords) in which the predicted trajectories for dynamic obstacles are divided, generates better approximations of the VOs for non-linear trajectories. Intuitively, by having more segments, we can have a better estimation for $\mathbf{v}_A^{opt}$, but because it gradually increases the complexity of the calculation, it is convenient to use a conservative approximation that does not compromise the average time per iteration. ***Fig. 3.9*** shows the behavior of the three performance metrics by altering the parameter $Segments$. Recall that in the base case we take into account 5 segments. The values in which the parameter $Segments$ was varied are the following: 1 (conventional VO); 2; 3; 4; 5 (base case); 6; 7; 8; 9; 10.

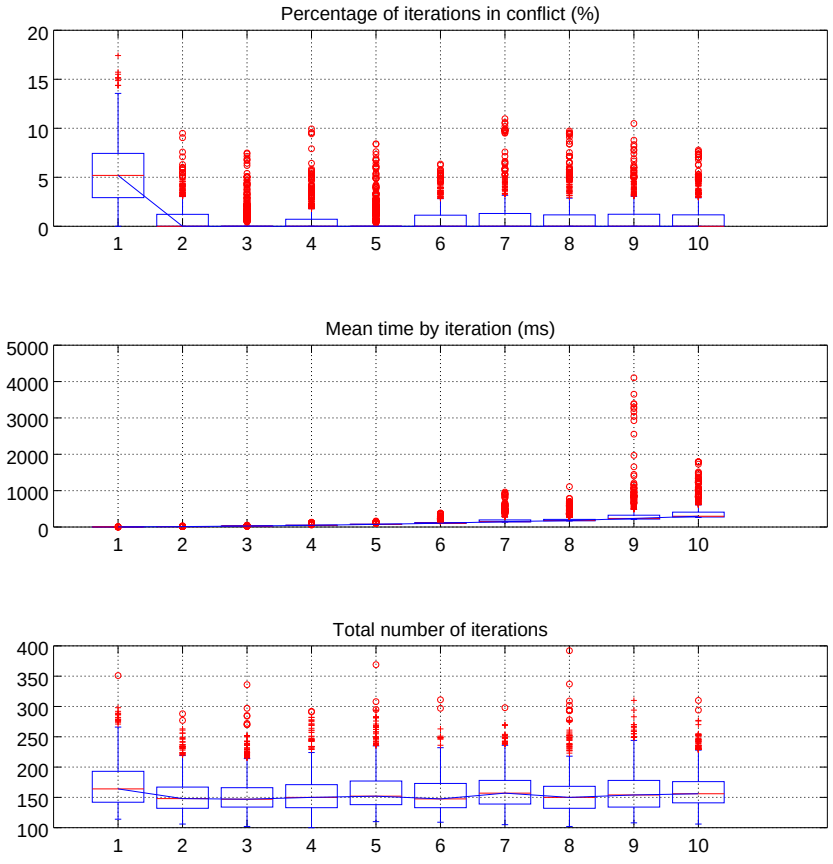


Figure 3.9: Performance metrics by varying the parameter $Segments$. The x axis shows the number of segments (chords) used for the predictions. With 5 in the base case.

With the help of ***Fig. 3.9 top***, we quickly see that the construction of a conventional VOs, i.e., in the case of a single segment, generates a considerably high percentage of iterations in conflict. The obtained mean is 5.38%, while from 2 segments, the average does not exceed 1%. We can also see that the best case, in this metric, is when we use 3 segments, followed by 5, with means of 0.37% and 0.5%, and values of Q_3 of 0%.

In ***Fig. 3.9 center***, we can see that for the base case, using a discretization of 6 segments is unsustainable based on Δt, because some of the outliers exceed that value. As expected, with a greater number of segments, the computational complexity increases, generating more VOs, more candidates and a greater number of steps to relax constraints. Therefore, the average time per iteration is increased.

Like the percentage of iterations in conflict, the total number of iterations, shown in ***Fig. 3.9 bottom***, increases when using the conventional VO, while using 2 or more segments, the quartiles Q_2 and Q_3 behave very similarly. In this case, no particular trend appears among outliers.

Variation of the parameter τ

The prediction duration (τ), is the time span from the current time in which the predicted trajectory will be traced. Intuitively, with a higher τ, the robot can better predict which velocities will be conflicting in the future. It may seem that, as with the parameter $nOPercentage$, the excessive increase of τ can be counterproductive, by saturating the ACV. However, due to the relaxation of constraints in time, and the fact that the set of furthest velocities produce VOs of smaller area, the increase of τ does not turn out to be counterproductive in this context. ***Fig. 3.10*** shows the behavior of the three performance metrics by altering the parameter τ. Recall that in the base case we take into account $5.0\ s$. The values in which the parameter was varied τ are the following: $2.5\ s$; $3.5\ s$; $5.0\ s$ (base case); $7.5\ s$; $10\ s$.

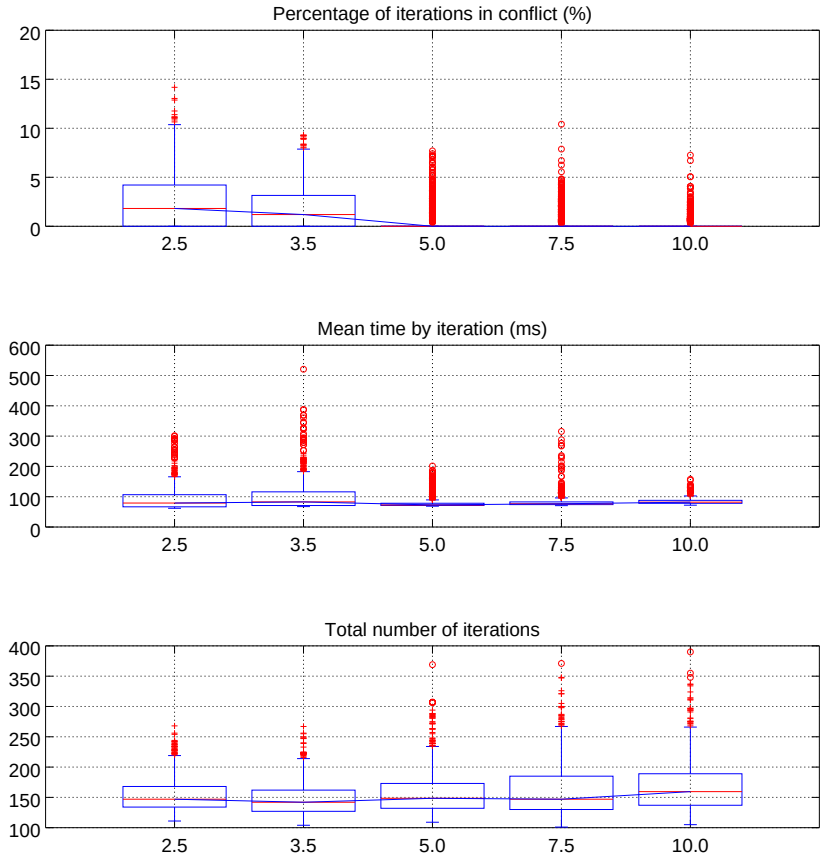


Figure 3.10: Performance metrics by varying the parameter τ. The x axis shows the duration of predictions in seconds. With $5.0\ s$ in the base case.

When increasing the parameter τ, we observe a decrease in the percentage of iterations in collision, in ***Fig. 3.10 top***. This behavior is intuitive, because the robot has a better notion of the future behavior of dynamic obstacles, which allows it to foresee probable collisions and avoid them with greater effectiveness.

In ***Fig. 3.10 center, bottom***, based on the average time per iteration and the total number of iterations, we can see that the best value assigned to τ for the experiments was of $10\ s$. While the last metric is less significant, the average time per iteration and the percentage of iterations in collision show as maximum outliers $157.7\ ms$ and 7.26%, respectively. The first is less than half of Δt, and the second one is the best *worst case*, in that metric varying τ.

Variation of the parameter Δt

It is possible that Δt is the most important parameter in the simulation, because it determines the *rate* at which the dynamic obstacles are *observed*, and at which the robot is controlled. The magnitude of Δt directly influences the dimensions of the ACV: like the parameter $RestAccScale$, the higher the value, the larger will be the ACV. ***Fig. 3.11*** shows the behavior of the three performance metrics by altering the parameter Δt. Recall that in the base case, we take into account $0.35\ s$. The values in which the parameter was varied Δt are the following: $0.1\ s$; $0.25\ s$; $0.35\ s$ (base case); $0.5\ s$; $0.75\ s$.

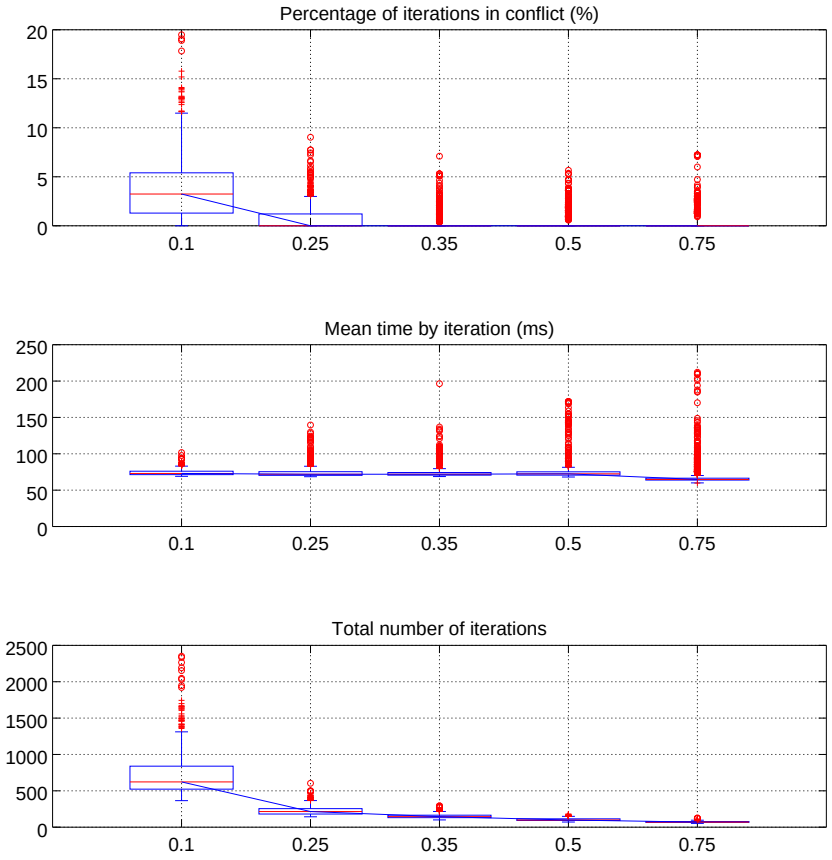


Figure 3.11: Performance metrics by varying the parameter Δt. The x axis shows the duration in seconds of this parameter. With $0.35\ s$ in the base case.

Fig. 3.11 bottom shows that the higher Δt, the lower the number of iterations to reach the *Goal*. This parameter influences the same way as $RestAccScale$, increasing the size

of the ACV. Recall that the bigger the ACV, the greater the search space for $\mathbf{v}_A^{opt}$. This allows that controls with speed equal or greater than the preferred speed may be used in greater proportion. Δt is a *dangerous* parameter since a very high configuration, i.e., a low frequency of updating of the controls, would cause an equally slow reaction capacity, and clumsy in cases where humans do not stick to predictions.

In ***Fig. 3.11 center***, we can observe that, in all cases of variation of this parameter, the maximum outliers are always below the value of Δt, that is, the average time per iteration is always acceptable.

Finally, in ***Fig. 3.11 top***, the effect of a smaller ACV is shown: the search space for the optimal velocity is smaller, so there may be no collision-free velocities, the robot takes a $zero$ velocity, and is hit by some dynamic obstacle. Recall that dynamic obstacles are *blind*.

Experiments with the Differential Drive Robot

As in the *holonomic* model, the algorithm was executed 8 thousand times with the base case setup, in order to obtain a sufficiently dense data set to evaluate the effect of the algorithm parameters. It is necessary to mention that the behavior of the algorithm in both models *(holonomic and DDR)*, have a very similar performance, according to the metrics.

Fig. 3.12 shows distributions similar to those in ***Fig. 3.4***. For example, at least 75% of the executions in the base case obtained 0% of iterations in collision.

In ***Fig. 3.12.center***, quartiles Q_1, Q_2 (median), Q_3, have values of $71.75, 73.25, 75.8$, respectively, while in the *holonomic* case they have values of $70.42, 71.96, 74.71$. Its mean has a value of $71.77\ (ms)$, and in the *holonomic* case $71.91\ (ms)$.

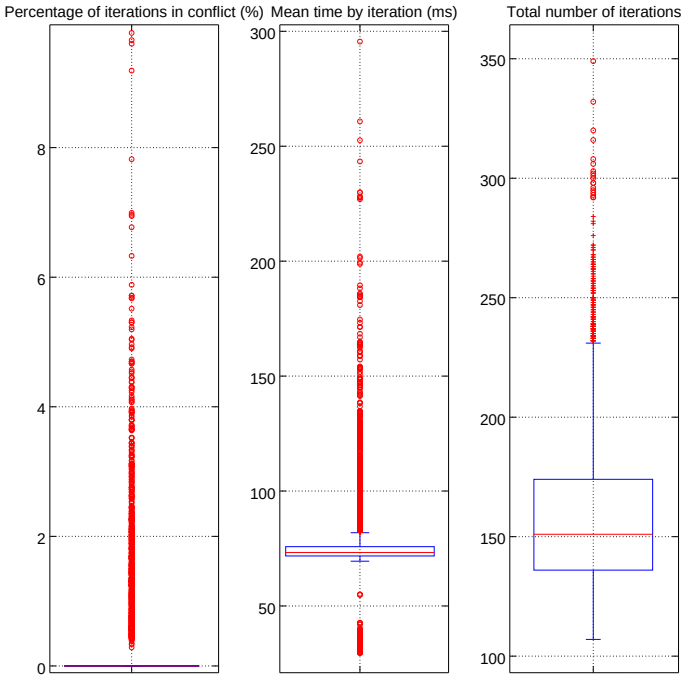


Figure 3.12: Three *boxplots* representing the distribution of the performance metrics proposed on the base case.

In ***Fig. 3.12.right***, quartiles Q_1, Q_2 (median), Q_3, have values of $136, 151, 174$, respectively, while in the *holonomic* case they show the values of $129, 145, 168$. The distribution is then similar in both cases, although the *DDR* case has higher quartiles, being approximately 5% biggers than the *holonomic* case.

Variation of the parameter nH

In ***Fig. 3.13***, we can see that the percentage of iterations with collision is distributed in a similar way in *holonomic* and in *DDR*, the latter being approximately 50% less conflictive, based on the outliers. We can also observe that the average times per iteration are almost the same, except for some *outliers*, as well as the total number of iterations.

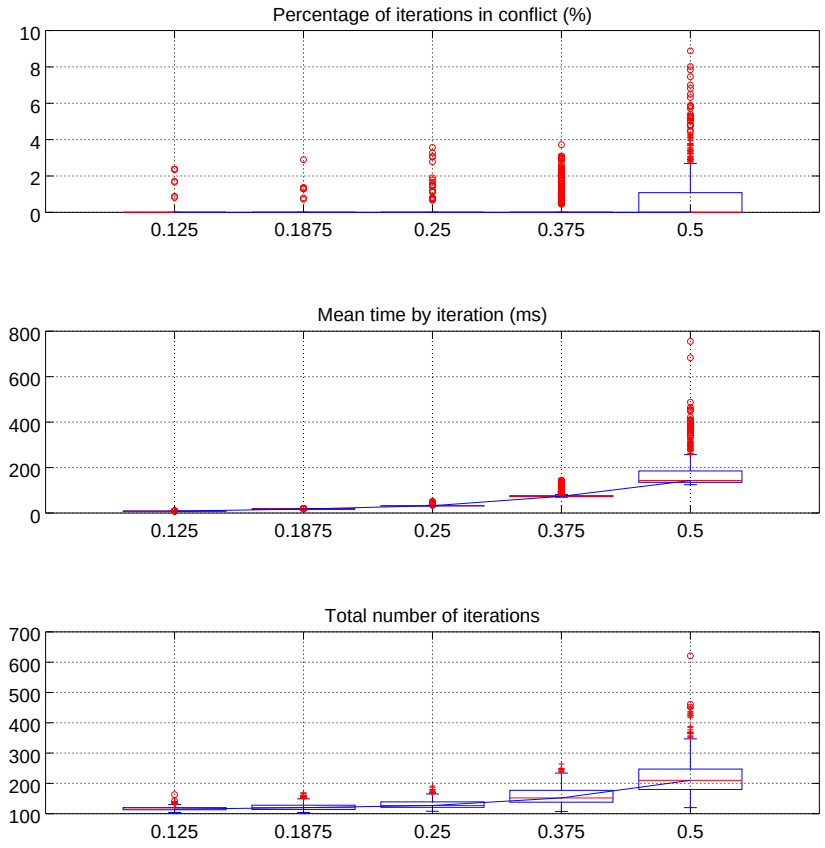


Figure 3.13: Performance metrics by varying the parameter nH for a DDR. The x axis shows the value of the human density of the environment, in $\frac{humans}{m^2}$. With $0.375\frac{humans}{m^2}$ in the base case.

Variation of the parameter nOPercentage

Fig. 3.14 shows that varying the parameter $nOPercentage$ does not give a clear trend in the percentage of iterations in conflict. However, the average time per iteration shows the same trend as with the holonomic robot: the higher the percentage in nO, the longer the iterations take. This parameter affects in the same proportion the holonomic case as the DDR.

As in the holonomic case, the total number of iterations does not show a clear trend when the parameter $nOPercentage$ is varied, but the value of both medians are very close to 150.

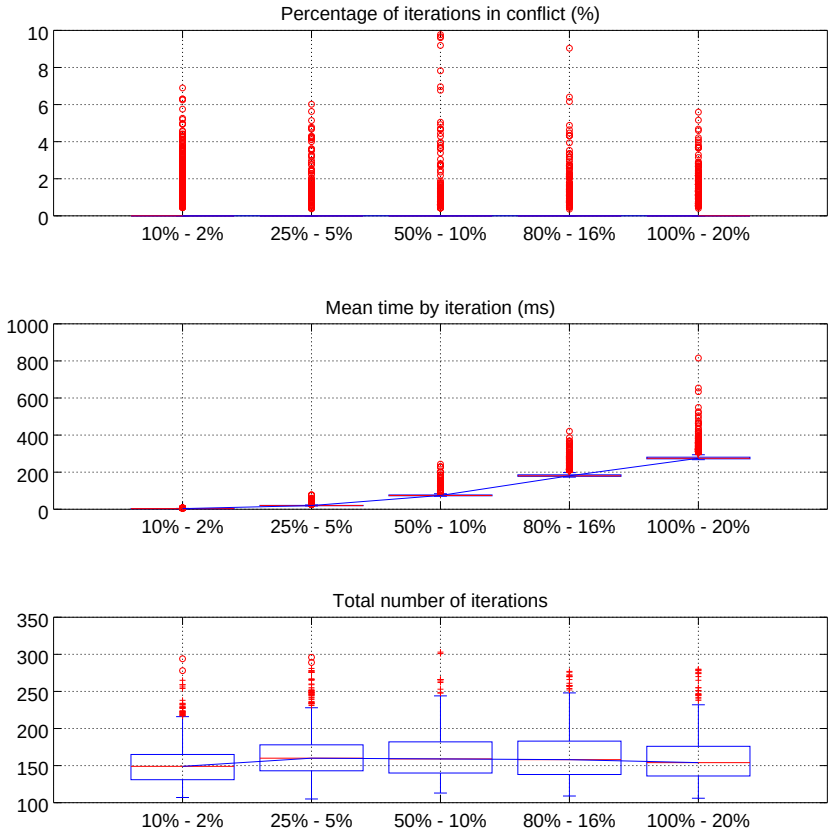


Figure 3.14: Performance metrics varying the parameter $nOPercentage$. The x axis of the three graphs shows the percentage that represents nO and nmO over the number of humans, (150). With $50\% - 10\%$ in the base case.

Variation of the parameter ComfortRadScale

Fig. 3.15 shows that, in general, the percentage of iterations in conflict is lower for the DDR case than for the holonomic case. The only trend that can be seen with the naked eye for the first metric, is that the higher $ComfortRadScale$, the lower the percentage of iterations in conflict, although the distribution of outliers tends to increase with a scale of 1.5.

The other two metrics show very similar distributions in both cases, with the exception of the outliers, which get to be of greater magnitude in the case of the DDR for the average time per iteration, but of smaller magnitude for the total number of iterations.

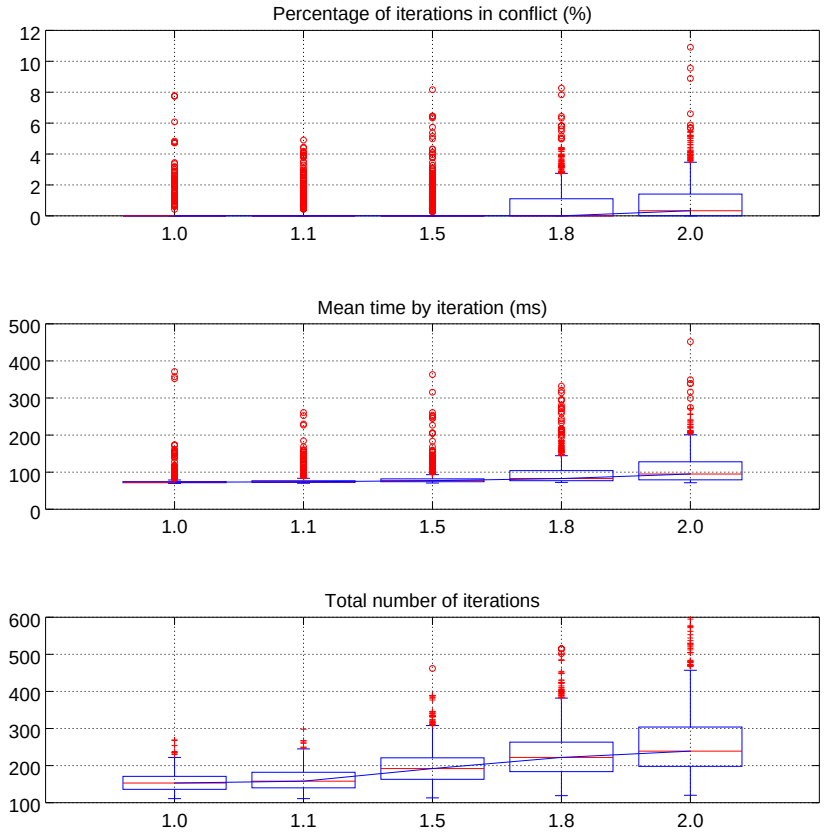


Figure 3.15: Performance metrics by varying the parameter $ComfortRadScale$ for a DDR. The x axis shows the scale of the comfort radius with respect to the radius of the robot. With 1.1 in the base case.

Variation of the parameter RestAccScale

As with the parameter $ComfortRadScale$, when viewing ***Fig. 3.16***, it can be deduced that, with an intermediate value of the parameter $RestAccScale$, it has a lower percentage of iterations in collision, based on the outliers, because the medians remain at 0%. Otherwise, in the holonomic case, the trend is: the higher $RestAccScale$, the lower the percentage of iterations in collision.

It can also be deduced that, compared to the $ComfortRadScale$ parameter, when $RestAccScale$ varies, not much impact is produced, exception made for the outliers. The effect for the DDR is to increase the average time per iteration, and decrease the total

number of iterations.

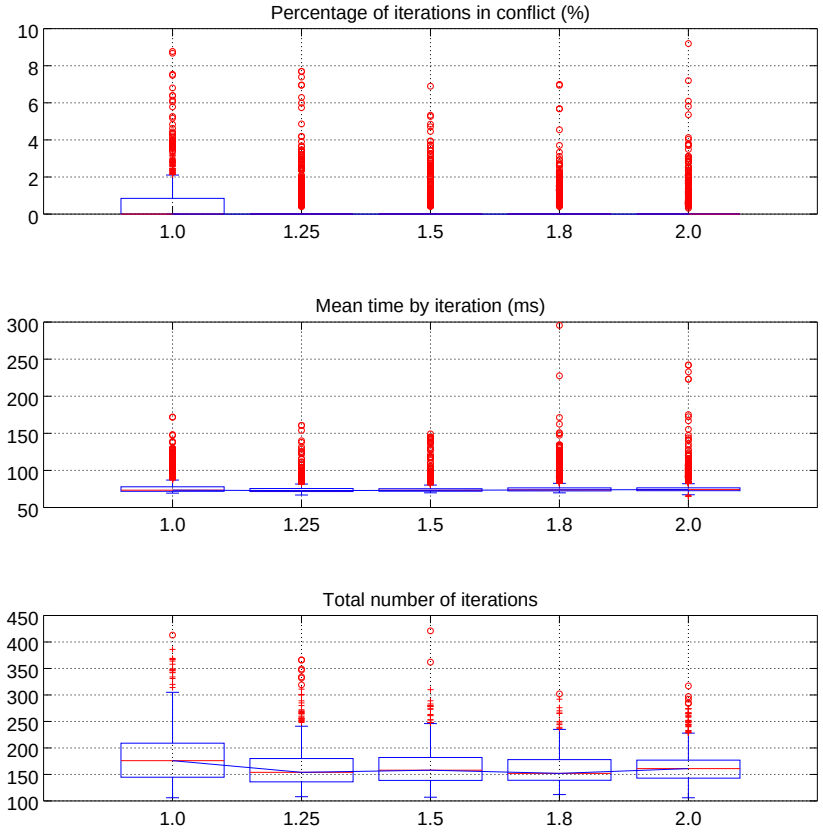


Figure 3.16: Performance metrics by varying the parameter $RestAccScale$ for a DDR. The x axis shows the scale of maximum accelerations for the robot with respect to its maximum control velocities. With 1.8 in the base case.

Variation of the parameter Segments

In ***Fig. 3.17***, we can see that the number of segments in which trajectories are discretized has the same effect in the holonomic case and in the DDR case. The distributions are practically the same: the greater the number of *segments*, the greater the average time per iteration. In the same way, the number of segments practically does not influence the total number of iterations necessary to reach the *Goal*.

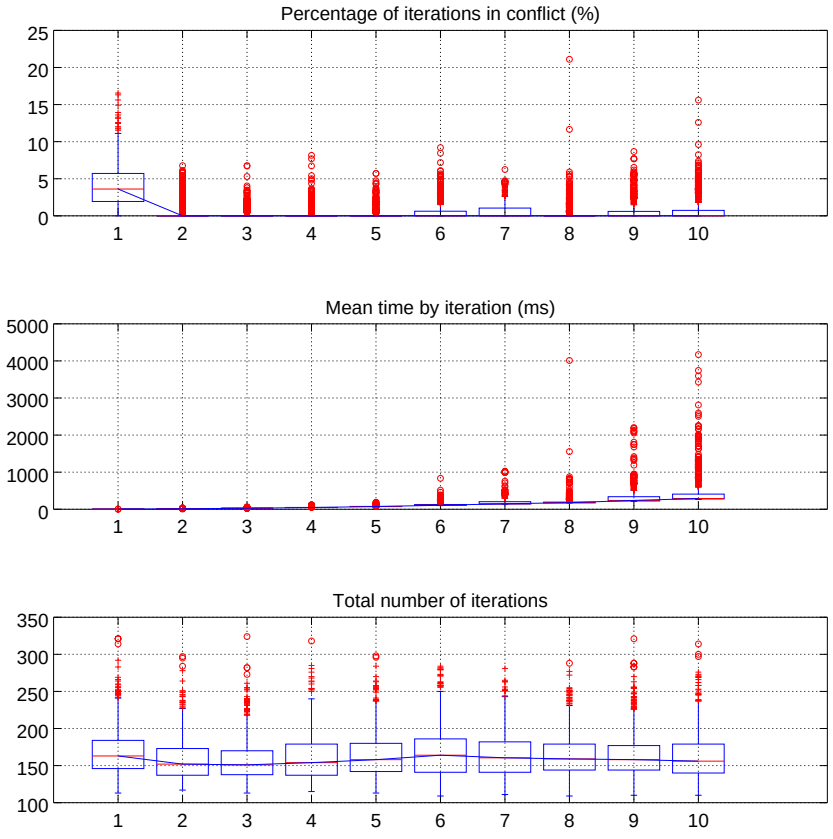


Figure 3.17: Performance metrics by varying the parameter $Segments$ for a DDR. The x axis shows the number of segments (chords) used for the predictions. With 5 in the base case.

Variation of the parameter τ

Performance metrics by varying the parameter τ, ***see Fig. 3.18***, generally show a lower percentage of conflicting iterations, having the same tendency as in the holonomic case. The average time per iteration also shows the same tendency in both cases: this metric is not altered much when increasing τ, and the data follow very similar distributions. The total number of iterations also has the same tendency in both cases: the median of each configuration is close to 150 iterations. The difference between both cases is the magnitude of the outliers, although it is not significant.

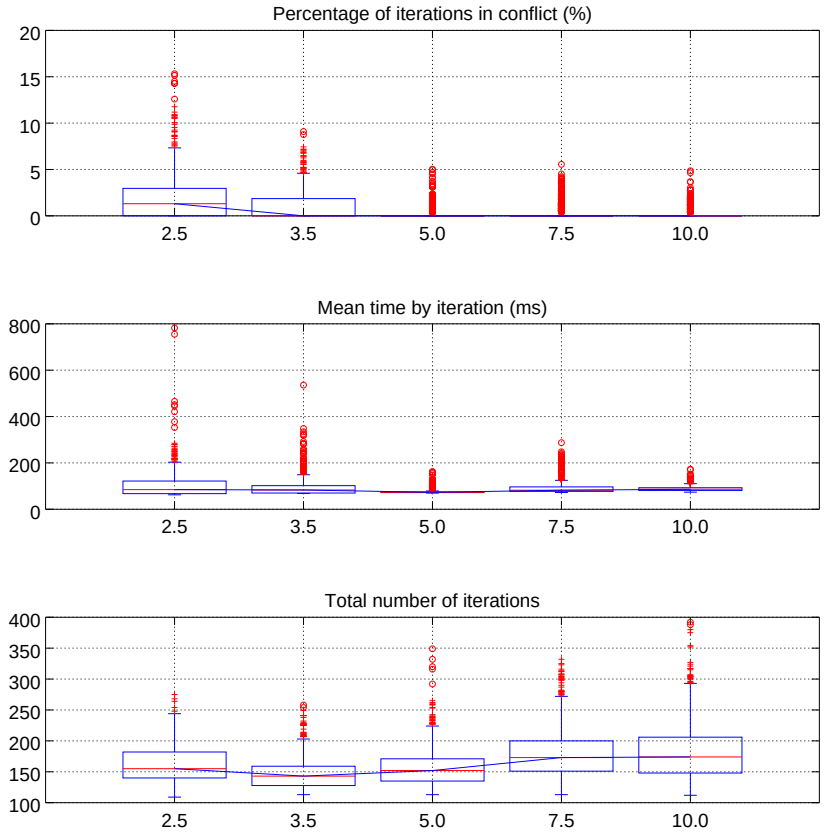


Figure 3.18: Performance metrics by varying the parameter τ for a DDR. The x axis shows the duration in seconds of predictions. With $5.0\ s$ in the base case.

Variation of the parameter Δt

The effect of the parameter Δt is shown in ***Fig. 3.19***. The percentage of iterations in conflict has the same tendency in the holonomic robot and the DDR when varying this parameter. In both cases, increasing this parameter reduces the value of the metric, but with $\Delta t = 0.75\ s$, in both cases the magnitude of the outliers is increased.

Both the average time per iteration and the total number of iterations to reach the *Goal*, show the same tendency in both cases. In the first of these metrics, the only apparent significant difference is in the magnitude of the outliers, while in the last one, even the outliers show a similar distribution.

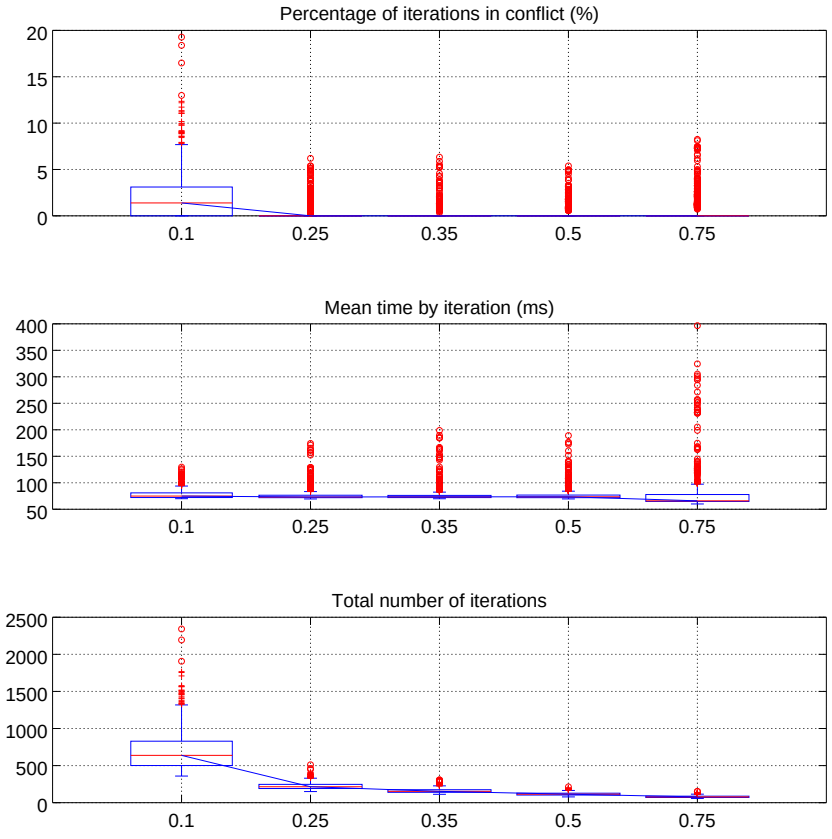


Figure 3.19: Performance metrics by varying the parameter Δt for a DDR. The x axis shows the duration in seconds of this parameter. With $0.35\ s$ in the base case.

Chapter 4

Probabilistic Approach

4.1 Introduction

Up to now, we have worked with deterministic predictions, i.e., the robots have the notion that the mobile obstacles will follow exactly the predicted trajectories. In the probabilistic modelling presented in this chapter, we suppose that we have richer information, namely a full distribution on the possible trajectories followed by the mobile agents. At some time t, we assume that we are able to maintain the prediction over one mobile obstacle trajectory. Let us index the mobile obstacles with $\mathcal{O}$. The model that we use for that has the following form, which is a sum over M different prediction models:

$$p(\mathbf{p}_{\mathcal{O}}(t)) = \sum_{m \in M} p(\mathbf{p}_{\mathcal{O}}(t)|m)p(m)$$

where $\mathbf{p}_{\mathcal{O}}(t)$ are potential trajectories of the mobile obstacle $\mathcal{O}$ at time t, and m indexes the different models. To simplify the use of this distribution, we obtain a series of trajectories, each represented by a set of points (x, y), whose uncertainty is represented by variances along each coordinate.

Now let us suppose a hypothetical velocity $\mathbf{v}_A(t)$ for the robot A, that will be taken at t and kept up to infinity. We want to know the probability that this velocity will end in a collision with one of the present agents, at some time $t' > t$. Hence, we want to determine the probability

$$p(c^{t,\infty} = 1|\mathbf{v}_A(t)).$$

Let us suppose that the robot is sharing its workspace with $n\mathcal{O}$ mobile obstacles. Then,

$$\begin{aligned} p(c^{t,\infty}=1|\mathbf{v}_A(t)) &= 1-p(c^{t,\infty}=0|\mathbf{v}_A(t)) \text{ (the complement event, i.e., that no collision will occur)} \\ &= 1-\prod_{o=1}^{n\mathcal{O}} p(c_o^{t,\infty}=0|\mathbf{v}_A(t)) \text{ (no collision with any of the mobile obstacles)} \\ &= 1-\prod_{o=1}^{n\mathcal{O}} (1-p(c_o^{t,\infty}=1|\mathbf{v}_A(t))) \end{aligned}$$

where $c^{t,\infty}=1$ is the event that there will be some collision between t and ∞, and $c_o^{t,\infty}=1$ is the event that there will be collision between the robot and the mobile obstacle indexed by o, between t and ∞. Now, by making the mobile obstacle velocity appear and by supposing (first) that the mobile obstacle maintains its velocity constant, we have

$$p(c_o^{t,\infty}=1|\mathbf{v}_A(t)) = \int_{\mathbf{v}_o} p(c_o^{t,\infty}=1|\mathbf{v}_A(t),\mathbf{v}_o)p(\mathbf{v}_o)d\mathbf{v}_o$$

where $p(c_o^{t,\infty}=1|\mathbf{v}_A(t),\mathbf{v}_o)$ is the probability of collision with mobile obstacle o, at some time after t, for a robot having velocity $\mathbf{v}_A(t)$ at t and keeping it, while o keeps its velocity $\mathbf{v}_o$. Now, in those conditions, the probability $p(c_o^{t,\infty}=1|\mathbf{v}_A(t),\mathbf{v}_o)$ is binary, depending on the value of $\mathbf{v}_A(t)$: 1 when $\mathbf{v}_o$ is inside a velocity obstacle generated by $\mathbf{v}_A(t)$, 0 otherwise; hence, we can write

$$p(c_o^{t,\infty}=1|\mathbf{v}_A(t)) = \int_{VO_{o|A}} p(\mathbf{v}_o)d\mathbf{v}_o,$$

where $VO_{o|A}$ is the velocity obstacle generated by the robot (at $\mathbf{p}_A(t)$, with velocity $\mathbf{v}_A(t)$) on the mobile obstacle o (at $\mathbf{p}_o$).

We end up with the following expression for the probability of collision with some mobile obstacle, for a robot maintaining its velocity at $\mathbf{v}_A(t)$, starting in t,

$$p(c^{t,\infty}=1|\mathbf{v}_A(t)) = 1-\prod_{o=1}^{n\mathcal{O}}(1-\int_{VO_{o|A}} p(\mathbf{v}_o)d\mathbf{v}_o).$$

Note that we can also write it:

$$1-p(c^{t,\infty}=1|\mathbf{v}_A(t)) = \prod_{o=1}^{n\mathcal{O}}(1-\int_{VO_{o|A}} p(\mathbf{v}_o)d\mathbf{v}_o) \tag{4.1}$$

and by taking the $\log$ of the previous expression

$$\log(1-p(c^{t,\infty}=1|\mathbf{v}_A(t))) = \sum_{o=1}^{n\mathcal{O}}\log(1-\int_{VO_{o|A}} p(\mathbf{v}_o)d\mathbf{v}_o). \tag{4.2}$$

This expression is obviously untractable, but here we give some hints on how to estimate this.

First of all, the robot velocities $\mathbf{v}_A(t)$ may be discretized in a regular grid. This grid should not be too large because we are only concerned by selecting velocities around the current robot velocity. We name this grid as $Grid_A(t)$, and it represents (in a discretized way) the ***Attainable Cartesian Velocities*** (ACV)[13], at current time t.

Furthermore,

- the time horizon can be set finite as τ and the reasoning will be the same as above;
- the terms $\int_{VO_{\mathcal{O}|A}} p(\mathbf{v}_{\mathcal{O}})d\mathbf{v}_{\mathcal{O}}$ can be computed fast, e.g., by sampling (Monte-Carlo).

4.2 Monte-Carlo implementation

A sampling method can give a relatively efficient way to approximate the integral $\int_{VO_{\mathcal{O}|A}} p(\mathbf{v}_{\mathcal{O}})d\mathbf{v}_{\mathcal{O}}$ of Eq. 4.2. For easiness in the explanations, let us first consider that only one mobile obstacle, indexed by $\mathcal{O}$, is being described by the distribution of its current velocity (not a whole predicted trajectory).

Refer to figure 4.1 below. The current robot position is represented by the point $\mathbf{p}_A$. The mobile obstacle position and velocity being uncertain, its distribution is represented by samples, indexed with indices between parenthesis. Given the (fixed) robot position $\mathbf{p}_A$, then each sample (j) of the mobile obstacle $\mathcal{O}$ position/velocity gives rise to a velocity obstacle defined by the mobile obstacle candidate velocity $\mathbf{v}_{\mathcal{O}(j)}$ and its position $\mathbf{p}_{\mathcal{O}(j)}$.

In Fig. 4.1, for one mobile obstacle $\mathcal{O}$ and for two samples (1) and (2), the corresponding velocity obstacle generated by the robot has been drawn. We focus on the holonomic case, so we depict the *ACV* as a rectangle. Note that each of the VOs generated on the samples allows to generate a forbidden region for the robot velocity where the translated versions of the VO may include the velocity $\mathbf{v}_{\mathcal{O}(i)}$. Those regions intersect the *ACV* along some shape (which has been drawn in green and red). Now, imagine that the *ACV* is regularly sampled. Note that this is a second sampling process among candidates velocities for the robot, represented as cells indexed by k in the grid $Grid_A(t)$.

Then, to evaluate $\int_{VO_{\mathcal{O}|A}} p(\mathbf{v}_{\mathcal{O}})d\mathbf{v}_{\mathcal{O}}$, we use a Monte Carlo approximation:

$$\int_{VO_{\mathcal{O}|A}} p(\mathbf{v}_{\mathcal{O}})d\mathbf{v}_{\mathcal{O}} \approx \frac{1}{N}\sum_{j=1}^{N} I_{VO_{\mathcal{O}(j)|A}}(\mathbf{v}_{\mathcal{O}(j)}) \tag{4.3}$$

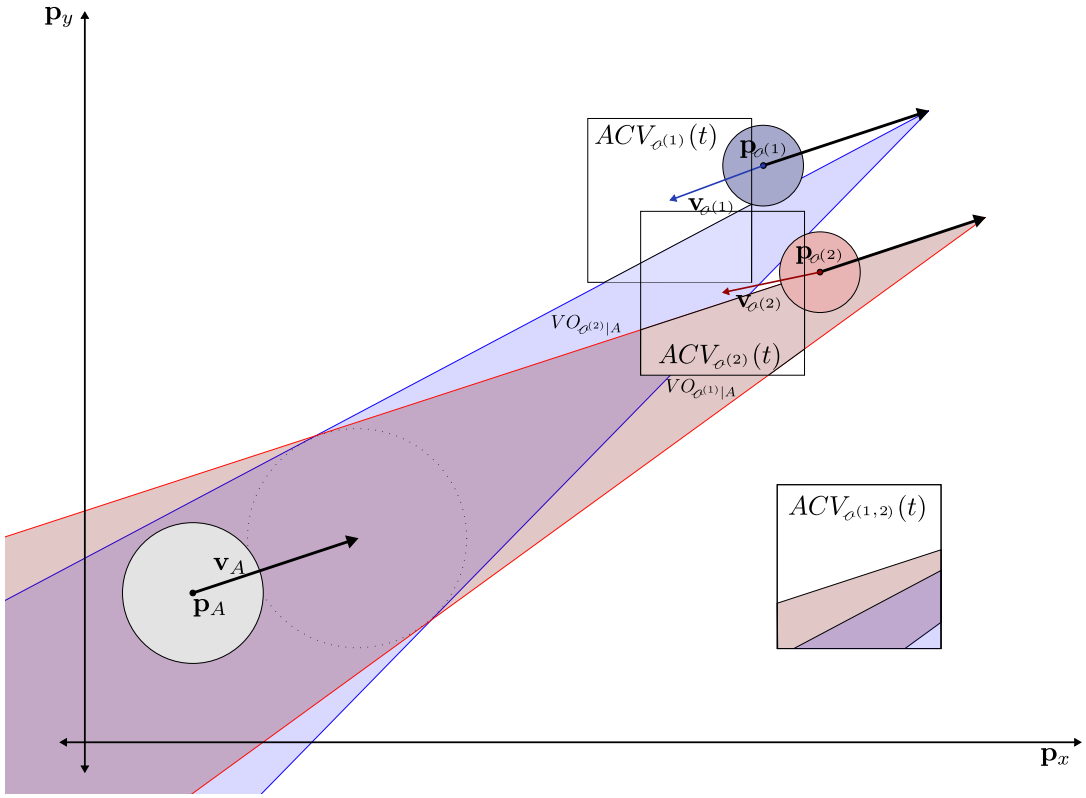


Figure 4.1: Sampling approach for the probabilistic version. $ACV_{\mathcal{O}^{(1,2)}}(t)$ corresponds to the ACV of $\mathcal{O}$, generated when processing these two samples, and assuming the same restrictions on velocity of A.

where N is the number of samples and $I_{VO_{\mathcal{O}^{(j)}|A}}(\mathbf{v}_{\mathcal{O}^{(j)}}) = 1$ if and only if $\mathbf{v}_{\mathcal{O}^{(j)}}$ is inside the $VO_{\mathcal{O}^{(j)}|A}$, 0 otherwise. With the geometric approach described above, this sum can be evaluated sample by sample: each sample is processed, and the colored region computed above will allow to update the cells $Grid_A^{(k)}(t)$ where a collision occurs. The indicators above are used to increment a counter associated to each of it. When all the N mobile obstacle velocity samples have been processed, we have at each cell $Grid_A^{(k)}(t)$ the estimate of $\frac{1}{N}\sum_{j=1}^{N} I_{VO_{\mathcal{O}^{(j)}|A}}(\mathbf{v}_{\mathcal{O}^{(j)}})$.

The geometric approach described above can be simplified as illustrated in Fig. 4.2: instead of considering the velocity obstacle generated by the robot on the mobile obstacle, one can see from the figure that on the reverse way, it is possible to reason on the velocity obstacles generated by each mobile obstacle sample on the robot. The principle is the same: each VO intersects the ACV along some shape; once this shape is determined, it becomes possible to iterate on the affected cells of $Grid_A(t)$ to update the corresponding probability of collision and to deduce the approximation of Eq. 4.3.

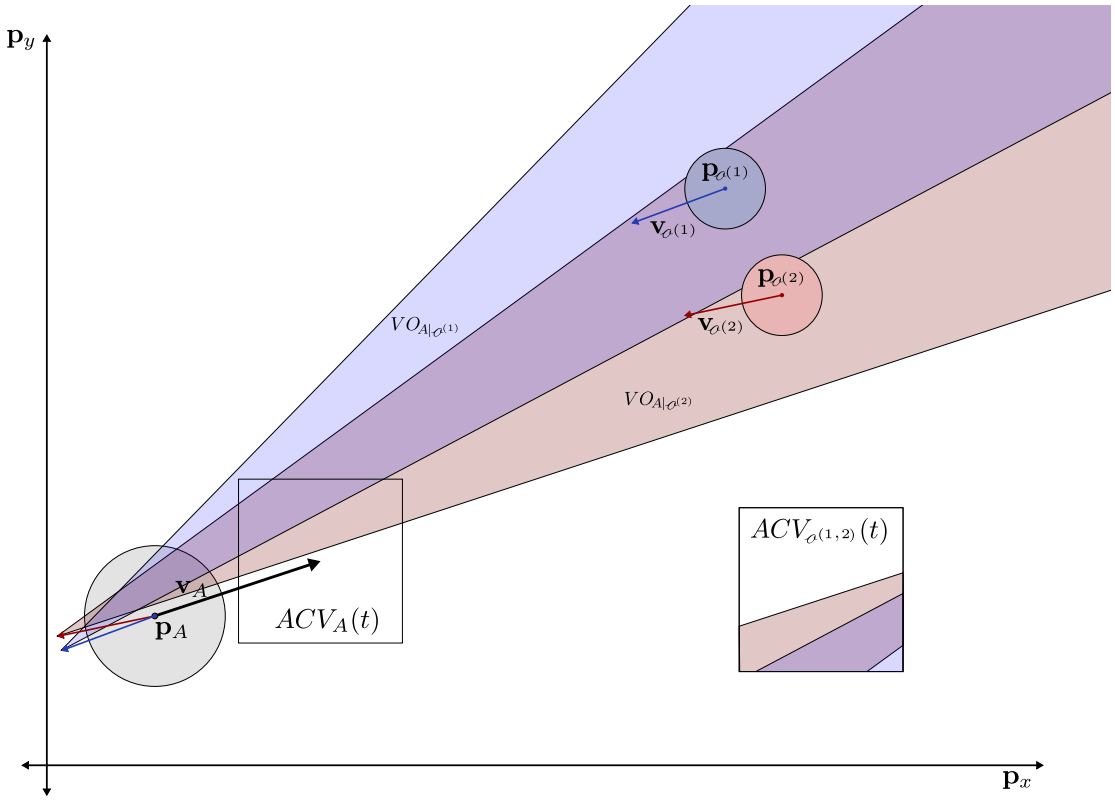


Figure 4.2: Simplified version of sampling. Note that $ACV_A(t)$ is the same as $ACV_{\mathcal{O}^{(1,2)}}(t)$ rotated by π rads. That is because $VO_{A|\mathcal{O}^{(j)}}$ and $VO_{\mathcal{O}^{(j)}|A}$ are the same, but in the opposite direction.

4.3 Extension to piecewise linear mobile obstacle trajectories

Starting from the explanations above, we can first generalize them to the case the velocities of the mobile obstacles are constant on intervals $[\tau_i, \tau_{i+1}]$ with $\tau_{i+1} > \tau_i$. The derivations are quite similar, and the only difference is that the velocity obstacle is a truncated cone delimited by arcs of circle on its extremities, as seen on *Chapter 1*. The figure 4.3 illustrates this principle for the case of two samples of a mobile obstacle position/velocity in $[\tau_i, \tau_{i+1}]$. Note that the underlying algorithm will allow to estimate

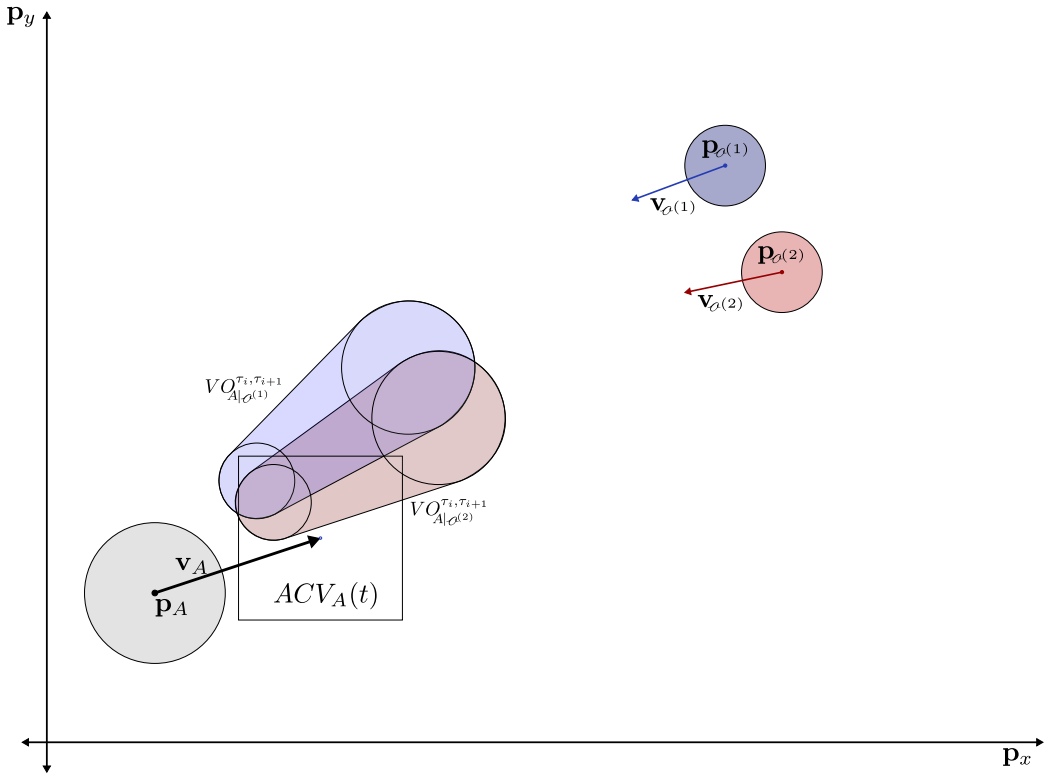


Figure 4.3: Extension to the update of the probabilities of collision on some interval $[\tau_i, \tau_{i+1}]$.

$$\begin{aligned} 1 - p(c^{[i]} = 1) &= \prod_{\mathcal{O}=1}^{n\mathcal{O}} (1 - \int_{VO^{\tau_i,\tau_{i+1}}_{\mathcal{O}|A}} p(\mathbf{v}_{\mathcal{O}}) d\mathbf{v}_{\mathcal{O}}) \\ &\approx \prod_{\mathcal{O}=1}^{n\mathcal{O}} (1 - \tfrac{1}{N} \textstyle\sum_{j=1}^{N} I_{VO^{\tau_i,\tau_{i+1}}_{\mathcal{O}^{(j)}|A}}(\mathbf{v}_{\mathcal{O}^{(j)}})) \\ &\approx \prod_{\mathcal{O}=1}^{n\mathcal{O}} (1 - \tfrac{1}{N} \textstyle\sum_{j=1}^{N} I_{VO^{\tau_i,\tau_{i+1}}_{A|\mathcal{O}^{(j)}}}(\mathbf{v}_A(t))) \end{aligned}$$

where $p(c^{[i]} = 1) \equiv p(c^{\tau_i,\tau_{i+1}} = 1|\mathbf{v}_A(t))$ for simplification, and $VO^{\tau_i,\tau_{i+1}}$ refers to the truncated velocity obstacle. Note that the position to take into account for this velocity obstacle is the one at $t + \tau_i$.

Given a linear trajectory of a mobile obstacle, discretized in $\mathfrak{z}$ segments, we can generalize the previous considerations for the two first time intervals, we have:

$$1 - p(c^{[1,2]} = 1) = p(c^{[1]} = 0) + p(c^{[2]} = 0) - p(c^{[1]} = 0) * p(c^{[2]} = 0),$$

then, for the first three time intervals:

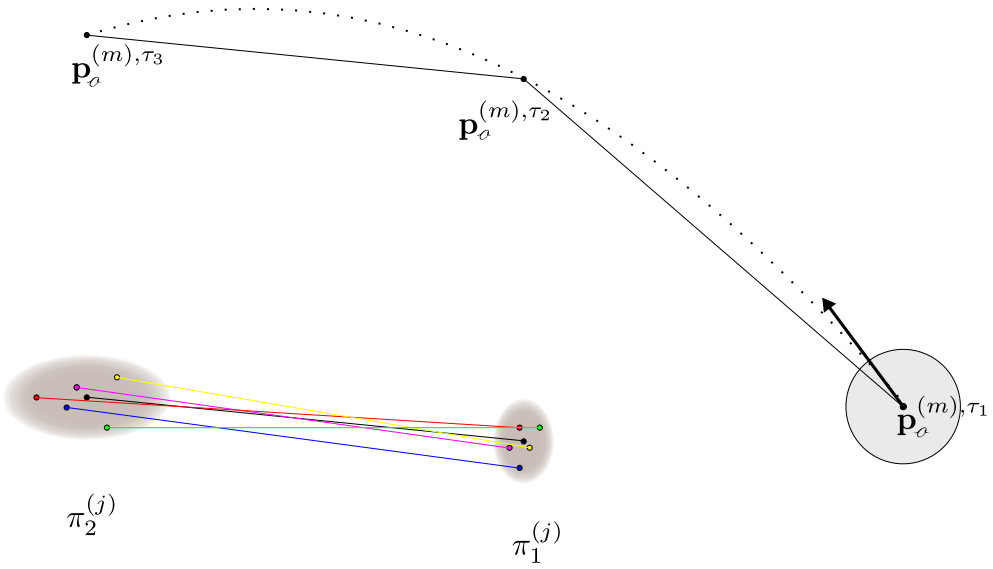


Figure 4.4: Top: one predicted trajectory model m, discretized in two segments between points $\mathbf{p}_o^{(m),\tau_1}$, $\mathbf{p}_o^{(m),\tau_2}$ and $\mathbf{p}_o^{(m),\tau_3}$. Bottom: in black, the second chord of the discretization. Five pairs of points π_1, π_2 sampled for the segment between times τ_2, and τ_3. The ellipses in gray represent the uncertainty in the position, described with Gaussian normal distributions in $2D$.

$$1 - p(c^{[1,3]} = 1) = p(c^{[1,2]} = 0) + p(c^{[3]} = 0) - p(c^{[1,2]} = 0) * p(c^{[3]} = 0),$$

and finally for all the $\jmath$ segments of the predicted trajectory:

$$1 - p(c^{[1,\jmath]} = 1) = p(c^{[1,\jmath-1]} = 0) + p(c^{[\jmath]} = 0) - p(c^{[i,\jmath-1]} = 0) * p(c^{[\jmath]} = 0).$$

Now, referring to equation 4.3, for a single obstacle o, and for each chord (*segment*) of one of its given prediction models m, each sample (j) is generated with the following steps: select two points, π_1, π_2, around the predicted points for the time span τ_i, τ_{i+1} in which the chord is traced, ($\mathbf{p}_o^{(m),\tau_i}$ and $\mathbf{p}_o^{(m),\tau_{i+1}}$), according to Gaussian distributions, with the points as means, and with their variances $(\sigma_{o_x}^{(m),\tau_i})^2$, $(\sigma_{o_y}^{(m),\tau_i})^2$, and $(\sigma_{o_x}^{(m),\tau_{i+1}})^2$, $(\sigma_{o_y}^{(m),\tau_{i+1}})^2$ associated to them. More formally, π_1, π_2 are sampled according to $\mathcal{N}(\mathbf{p}_o^{(m),\tau_i}, ((\sigma_{o_x}^{(m),\tau_i})^2, (\sigma_{o_y}^{(m),\tau_i})^2))$ and, $\mathcal{N}(\mathbf{p}_o^{(m),\tau_{i+1}}, ((\sigma_{o_x}^{(m),\tau_{i+1}})^2, (\sigma_{o_y}^{(m),\tau_{i+1}})^2))$, respectively. ***See fig. 4.4 bottom***.

All of the above holds for a single trajectory model, with different segments and with uncertainty in $\mathbf{p}_o(t)$ and $\mathbf{v}_o(t)$. Suppose now that we have M predicted trajectory models, each with an associated occurrence probability. How to compute the grid while taking

into account these models? The procedure can be done in a relatively simple way: do exactly the above, but for each sample, randomly choose a model m, according to the probability associated with it. The complete procedure to obtain $\mathbf{v}_A^{opt}$ is shown in the following section.

4.4 Proposed algorithm

The algorithm to find $\mathbf{v}_A^{opt}$ in the probabilistic approach is very different from the deterministic one. In this approach enter new notions such as the discretization of the robot velocities in a grid and the use of a $threshold$, which refers to the maximum collision probability value that we are willing to take when choosing the optimal velocity. Also, the way we select the best candidate velocity is through a *BFS, (Breadth-first search)* [24], starting in the nearest cell, or the cell containing $\mathbf{v}_A^{pref}$, and ending whenever a probability inferior to $threshold$ is found.

The algorithm is divided into two large parts: the first, contemplated in ***lines 1 - 13***, consists of filling $Grid_A(t)$ with the estimated probability of collision, induced by the trajectory prediction models of the nO obstacles closest to the robot. The second part of the algorithm, ***lines 14 - 24***, consists of choosing the optimal velocity for the robot, considering the constraints on the velocity space.

Below is the algorithm and the description of each operation. As input parameters we have the following:

τ: Duration of predictions.

$\mathfrak{z}$: Number of segments into which the predicted trajectories are divided.

$maxVel_A$: Maximum attainable velocity by robot A.

$threshold$: the maximum collision probability value that we are willing to take when choosing $\mathbf{v}_A^{opt}$.

N: Number of samples to estimate $\int_{VO_{o|A}^{\tau_i,\tau_{i+1}}} p(\mathbf{v}_o)d\mathbf{v}_o$.

$cols$, $rows$: Dimensions of $Grid_A(t)$.

nO: value of the maximum number of obstacles that the robot can *observe*, that is, for which the robot can estimate their positions and velocities. This value is set depending on the memory and computational capacity available.

$\mathcal{O}$: The set of obstacles visible by the robot, where each $o \in \mathcal{O}$ has its own trajectory prediction models within a time horizon τ.

Algorithm 2 Algorithm for choosing the optimal velocity

Input: τ, $\mathfrak{z}$, t_c, $maxVel_A$, $threshold$, N, $cols$, $rows$, nO, $\mathcal{O}$
Output: $\mathbf{v}_A^{opt}$

compute $ACV_A(t)$ and discretize it into $Grid_A(t)$, a matrix of $cols \times rows$
build the $\mathfrak{z}$ time span grids $TSG_A(t)$, matrices of $cols \times rows$
for all the nearest nO mobile obstacles in $\mathcal{O}$, indexed by $\mathfrak{o}$ **do**
  Receive the M trajectory prediction models, representing the next time span τ
  for j = 1 $\rightarrow$ N **do**
    Randomly select from M a model m
    for i = 1 $\rightarrow$ $\mathfrak{z}$ **do**
      Generate sample points and estimate a position and velocity of $\mathfrak{o}$: $\mathbf{p}_{\mathfrak{o}^{(j)}}^{(m),\tau_i}$, $\mathbf{v}_{\mathfrak{o}^{(j)}}^{(m),\tau_i}$
      Generate the velocity obstacle $VO_{A|\mathfrak{o}^{(j)}}^{\tau_i,\tau_{i+1}}$
      if distance($VO_{A|\mathfrak{o}^{(j)}}^{\tau_i,\tau_{i+1}}$, $\mathbf{v}_A(t)$) $< \|maxVel_A\| * t_c$ **then**
        Intersect $VO_{A|\mathfrak{o}^{(j)}}^{\tau_i,\tau_{i+1}}$ with the $ACV_A(t)$ and deduce the updatable cells (k)
        For all the updatable cells, update $TSG_A^{i,(k)}(t)$ with $\frac{1}{N}\sum_{j=1}^{N} I_{VO_{A|\mathfrak{o}^{(j)}}^{\tau_i,\tau_{i+1}}}(\mathbf{v}_A^{(k)}(t))$
  Update the value of $1 - \prod_{\mathfrak{o}=1}^{n\mathcal{O}} (1 - \frac{1}{N}\sum_{j=1}^{N} I_{VO_{A|\mathfrak{o}^{(j)}}^{\tau}}(\mathbf{v}_A(t)))$ on $Grid_A(t)$, with the contribution of $\mathfrak{o}$ on each $TSG_A(t)$
calculate the preferred velocity $\mathbf{v}_A^{pref}$
calculate the cell corresponding to $\mathbf{v}_A^{pref}$, $Grid_A^{(pref)}(t)$
repeat
  $resultCell = BFS(Grid_A^{(pref)}(t), threshold)$
  if $resultCell$ is valid **then**
    return convToVel($resultCell$)
  else
    Subtract from $Grid_A(t)$ the contribution of $TSG_A^{\mathfrak{z}}(t)$
    $\mathfrak{z} = \mathfrak{z} - 1$
until $\mathfrak{z} \leq 1$
return the global minimum found with BFS

The algorithm is detailed below. In ***line 1***, we compute $ACV_A(t)$, as described in previous chapters. Then, $Grid_A(t)$ is built as a matrix of $cols \times rows$. This matrix stores the probability of collision for the speeds achievable by the robot. The greater the number of cells, the greater the precision of the calculation. It is initialized with all its cells in 0, because it represents the probability of collision with any of the obstacles, i.e., $p(c^{[1,\mathcal{J}]} = 1)$, before considering any of them.

In ***line 2***, we build $\mathcal{J}$ grids, similar to $Grid_A(t)$, which represents the contribution of the nO nearest obstacles in the $\mathcal{J}$ time lapses. We call these grids $TSG^i_A(t)$, where the index i, represents the $i-th$ segment.

The code between ***lines 3 - 13*** fills $Grid_A(t)$ with the probability of collision, deduced from the predictions for the nO obstacles. In ***line 3***, one loop is started to iterate over the nO obstacles closest to the robot A. In ***line 4***, the data corresponding to the M trajectory prediction models, for the time period between $[t, t+\tau]$ are received and processed.

To sample the VOs, a loop *for* $j = 1 \rightarrow N$ is initiated. The first instruction in the loop is to randomly select a model m from the M received. The selection is made according to the probability of occurrence associated to them, i.e., the higher it is, the greater the probability of being chosen.

Lines 7 - 12 result in the contribution of the selected model m for the obstacle $\mathcal{o}$, over each time lapse indexed by i. For each of them, a probability of collision, induced by $\mathcal{o}$, accumulates in $TSG^i_A(t)$. For this, we first estimate a position and velocity, deduced from two points π_1, π_2, as mentioned above, ***see Fig. 4.4***. Once we have $\mathbf{p}^{(m),\tau_i}_{\mathcal{o}(j)}$, $\mathbf{v}^{(m),\tau_i}_{\mathcal{o}(j)}$, we deduce the VO (***line 9***).

In ***line 10***, we do a *quick test* to rule out generated VOs that do not have influence on the $ACV_A(t)$. The test is not completely accurate, since it considers the distance to the VO from its centroid, and not from its closest point to $\mathbf{p}_A$. In this way, we obtain a reduced set of VO, which leads to a decrement of the number of calculations of intersection with $ACV_A(t)$. After passing the *test*, in ***line 11***, we intersect the VO with the $ACV_A(t)$ to deduce the set of cells, indexed by k, within $TSG^i_A(t)$ that will be affected. In ***line 12*** we add on each affected cell $TSG^{i,(k)}_A(t)$ the value $\frac{1}{N}\sum_{j=1}^{N} I_{VO_{A|\mathcal{o}(j)}}(\mathbf{v}^{(k)}_A(t))$.

Once we have the distributions of probability of collision with $\mathcal{o}$, for each of the $\mathcal{J}$ segments, represented in the $TSGs$, we update on ***line 13*** the value of $1 - \prod_{\mathcal{o}=1}^{n\mathcal{O}}(1 - \frac{1}{N}\sum_{j=1}^{N} I_{VO_{A|\mathcal{o}(j)}}(\mathbf{v}_A(t)))$, i.e, $p(c^{[1,\mathcal{J}]} = 1)$ on $Grid_A(t)$, with the contribution induced by the obstacle $\mathcal{o}$.

Finally, to return $\mathbf{v}_A^{opt}$, we first calculate $\mathbf{v}_A^{pref}$ in ***line 14***, i.e., the velocity with magnitude s_{pref} and in direction to $\mathbf{p}_A^{Goal}$. Once calculated, we must now find $Grid_A^{(pref)}(t)$, i.e., the cell within $Grid_A(t)$ closest to $\mathbf{v}_A^{pref}$, or containing it.

In ***lines 16 - 23***, we perform a BFS over $Grid_A(t)$, starting from $Grid_A^{(pref)}(t)$, and we return the velocity corresponding to the center of the first cell found with $p(c^{[1,j]} = 1) \leq threshold$.

If a cell that meets this condition is not found, we relax the constraints in time (***line 21***), i.e., we remove the contribution to the probability of collision that generates the segment of the furthest trajectory in the prediction model. This process is repeated as long as the number of segments is strictly greater than 1.

In the case that we relax the restrictions until we consider only the first time lapse, and we can not find a cell with a value less than or equal to $threshold$ in $Grid_A(t)$, we return in ***line 24*** the velocity corresponding to the center of the cell with the minimum value. In case of a tie, we return the velocity corresponding to the center of the cell closest to $\mathbf{v}_A^{pref}$, i.e., the first one found by the BFS.

Chapter 5

Implementation and results of Probabilistic Approach

5.1 Implementation

As data input for the probabilistic model implementation, we use the same plain text format described in the chapter 3. In this version, we use either all the prediction models received, or only those with the higher probability of occurrence, by choice in the parameters, as opposed to the deterministic model, that only uses the most probable model. Each model m within the trajectory prediction mixture model for an obstacle $\mathcal{O}$, with duration τ, is described as a sequence of points $\mathbf{p}_{\mathcal{O}}^{(m)}$, with some uncertainty in their position, represented by the variances $(\sigma_{\mathcal{O}_y}^{(m)})^2$, and $(\sigma_{\mathcal{O}_x}^{(m)})^2$, respectively.

One of the main differences with respect to the deterministic approach is that we use now a grid to represent the ACV, which we name $Grid_A(t)$. The size of the search space for $\mathbf{v}_A$ will depend directly on the number of rows and columns of this discretized ACV. The more cells we represent in this grid, the more accurate and computationally expensive the calculation of $\mathbf{v}_A^{opt}$ will be. We use the QVO approach to represent the VOs, just like in the deterministic model (chapters 2 and 3).

As in the deterministic model, we use chords to discretize the predicted trajectories. Having uncertainty in both the velocity and the position of $\mathcal{O}$ over time, we use sampling, as mentioned in the previous chapter, to estimate probabilities of collision.

Sampling

To build the N velocity obstacles, for the chord (time segment) between the times τ_i, τ_{i+1}, we draw two points π_1, π_2, from the distributions $\mathcal{N}(\mathbf{p}_{\mathcal{O}}^{(m),\tau_i}, ((\sigma_{\mathcal{O}_x}^{(m),\tau_i})^2, (\sigma_{\mathcal{O}_y}^{(m),\tau_i})^2))$ and,

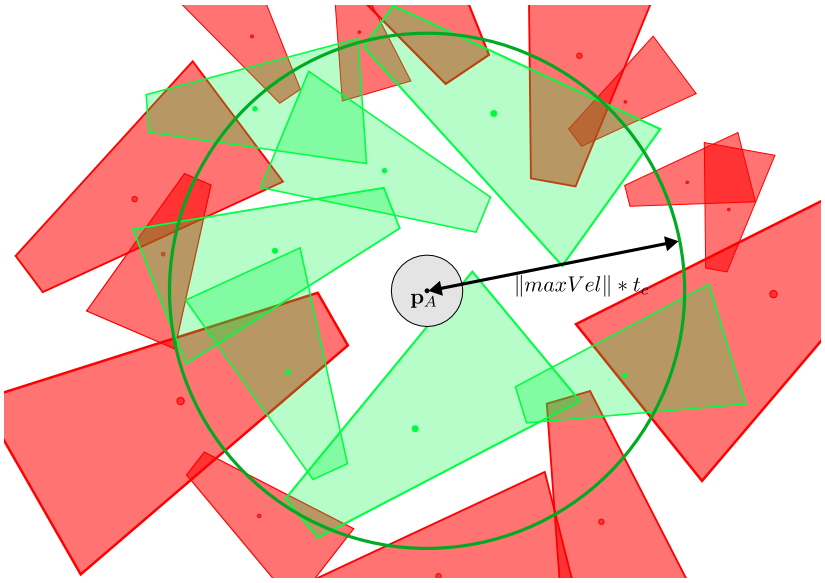


Figure 5.1: The VOs whose centroid is outside the green circle of magnitude $\|maxVel\|*t_c$, will not be taken into account to check their contribution to the ACV.

$\mathcal{N}(\mathbf{p}_{\mathscr{o}}^{(m),\tau_{i+1}}, ((\sigma_{\mathscr{o}_x}^{(m),\tau_{i+1}})^2, (\sigma_{\mathscr{o}_y}^{(m),\tau_{i+1}})^2))$, respectively. Then we can easily estimate the velocity of the *virtual obstacle*, $\mathbf{v}_{\mathscr{o}}^{(m)}(t) = \frac{\pi_2 - \pi_1}{\tau_{i+1} - \tau_i}$, and its position, $\mathbf{p}_{\mathscr{o}}^{(m)}(t) = \pi_1 - \mathbf{v}_{\mathscr{o}}^{(m)}(t) * \tau_i$, and build a velocity obstacle. As explained in the previous chapter, each of these velocity obstacles contributes $\frac{1}{N}$ to the probability of collision for $\mathscr{o}$, over the cells of $TSG_A^i(t)$ that intersect with them.

Because N samples are drawn per chord, the number of generated velocity obstacles may be very large, so it is necessary to do a *quick test* to discard samples without influence, as mentioned in ***line 14*** of ***Algorithm 2***, to process the intersections of a smaller number of velocity obstacles with the ACV. The test consists of drawing a circle centered on $\mathbf{p}_A$, of radius $\|maxVel_A\|*t_c$. This is a conservative way to take into account the reachable velocity obstacles, by considering the maximum speed of the robot A, in any direction, during the time t_c. As mentioned in the previous chapter, this method is not entirely accurate, but it does give a *quick and light* notion of which VOs are worth taking into account. ***See Fig. 5.1***.

The process to deduce which cells of $TSG_A^i(t)$ will be affected by each velocity obstacle, consists of the following steps:

1: Define a segment of horizontal line that we call *level*; at the beginning, *level* will cross through the center of all cells in the first row of $Grid_A(t)$.

2: Find the start and end indices, on the columns that contain some portion of the QVO over the current row, i.e., where it intersects any of the segments that delimit QVO,

or with the edges of the grid.

3: Store the indices of the columns start and end, and the index of the row.

4: Increase the *level* line segment to center it on the next row.

5: Repeat the steps $2-4$, until finishing with the rows.

To add the contribution of each obstacle to the probability of collision, first, all the cells of each $TSG^i_A(t)$ are initialized with a probability of collision 0.0, i.e., $p(c^{[i]}=1)=0$. For each sampling process, we use a temporary grid, which we call $tmpGrid$, also initialized with values in 0.0. After each sample is drawn, each affected cell is updated, adding the value of $\frac{1}{N}$ to it. Once we finish the sampling process, we update $TSG^i_A(t)$ with the probability of collision with both the current obstacle, or with any of the obstacles seen before, as described below. Let a be the probability of collision with the obstacle $\mathscr{o}_1$, and b the probability of collision with another obstacle $\mathscr{o}_2$, then the probability collision with either of the two obstacles is defined as $p(a \vee b) = p(a)+p(b)-p(a \wedge b)$. Being non-exclusive events, there may be cases in which the choice of a speed leads to collision with both at the same time. Here, $p(a \wedge b)$ is the probability that both events occur, and in this case it is given by $p(a \wedge b) = p(a) * p(b)$. Then, to add the contribution, we just update the value at each cell k in the following way:

$$TSG^{i,(k)}_A(t) = TSG^{i,(k)}_A(t) + tmpGrid^{(k)} - TSG^{i,(k)}_A(t) * tmpGrid^{(k)}.$$

Once the process of adding the contribution of each obstacle in the different $TSGs$ is finished, the probability of collision is updated in a similar way, by adding the contribution of all the $TSG^i_A(t)$ to $Grid_A(t)$:

$$Grid^{(k)}_A(t) = Grid^{(k)}_A(t) + TSG^{i,(k)}_A(t) - Grid^{(k)}_A(t) * TSG^{i,(k)}_A(t).$$

When necessary, we relax the constraints in time. This is done (as in the deterministic case) whenever no cell representing a velocity with a collision probability less than or equal to $threshold$ can be found. To do this, we withdraw the contribution of a $TSG^i_A(t)$ in $Grid_A(t)$. For each cell k, we apply the following:

$$Grid^{(k)}_A(t) = \frac{Grid^{(k)}_A(t) - TSG^{i,(k)}_A(t)}{1 - TSG^{i,(k)}_A(t)}.$$

In this case, we must validate the denominator to avoid numerical errors, i.e., when $1 - TSG^{i,(k)}_A(t) \approx 0$.

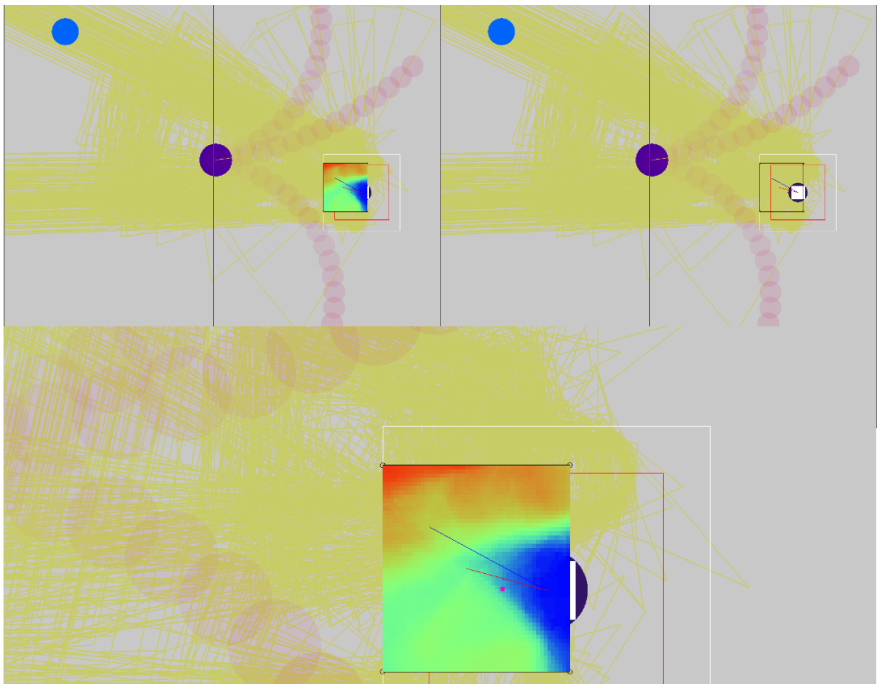


Figure 5.2: Above: a robot and a human, *left* visualization of $Grid_A(t)$ with temperature gradient colors, *right* visualization without $Grid_A(t)$. Bottom: Zoom on the grid. In magenta, we depict $\mathbf{v}_A^{opt}$ chosen for the next iteration.

5.2 Experiments with *Simulator*

As in the deterministic model, the visualization of the experiments was carried out using *OpenFrameworks development toolkit*. ***Fig. 5.2*** shows a visualization example. In this case, only one human is considered, with three trajectory prediction models, which consist of a series of points in the cartesian plane, and are shown as positions with magenta circles. The velocity obstacles generated in the sampling are represented as polygons of yellow outline and without filling. The grid that discretizes the *ACV* is represented in temperature color scale, i.e., the greater the probability of collision, the bluer is the color; on the contrary, the higher the collision probability, the redder is the color.

In ***fig. 5.2***, $\mathbf{v}_A^{pref}$ is represented by a blue line starting from the center of the robot. In red, one line represents $\mathbf{v}_A$. Finally, in magenta, we depict $\mathbf{v}_A^{opt}$ chosen for the next iteration, i.e., the velocity with probability less than or equal to $threshold$, 0.2 in this case, closest to $\mathbf{v}_A^{pref}$.

The experiments with the probabilistic model were carried out with holonomic robots, and the extension to differential drive robots remains as future work.

Setup of the experiments

In order to measure the performance of the algorithm, as in the deterministic model, we define a *base case*. Since the robots and the experiments are similar, we use the same values in the configuration parameters shared by the two models (see Chapter 3). To achieve a fair comparison, the extra parameters for this model were adjusted to obtain a similar average computational time per iteration as in the deterministic case. We established a maximum time of 2000 seconds to reach the goal, and in case of exceeding this time, the experiment is considered finished.

Recall that the experiments were performed in a simulated environment consisting of a space of squared shape and dimensions $400m^2$, where a set of randomly arranged humans move. The robot is placed on the circle inscribed to the environment, looking towards the center of it. The goal position $\mathbf{p}_A^{Goal}$ is also on the circle, in a position diametrically opposite to the initial position of A. Humans were simulated with prediction mixture models involving three models, with associated occurrence probabilities of $0.\bar{3}$, $0.\bar{2}$ and $0.\bar{2}$. The only shared parameter that was not configured with the same value as in the deterministic model is the number of humans. When using three models per human, the density of velocity obstacles is three times higher, therefore, to have a fair comparison, we use about $\frac{1}{3}$ of the humans in each configuration.

The following list shows the extra configuration parameters that were used for the experiments and their *default* value (base case):

nH: the number of humans in the environment, *(50 humans)*.

$rows$, $columns$: The dimensions of the grid. *(*140 *and* 140*, resulting on* 19600 *cells)*

$samples$: The number of samples to estimate the probability of collision induced by a velocity obstacle, *(100)*.

$threshold$: The maximum collision probability value that we are willing to accept when choosing the optimal velocity, *(0.2)*.

$timeToCollision$: The time we consider to perform the discard test, *(2.5)*.

In order to compare the performance of the robot with different setups, the experiments were performed by varying each of the parameters mentioned above, in addition to the parameters common with the deterministic case, in at least five different configurations. We use the same three metrics to measure the overall performance of the algorithm: *Percentage of iterations in conflict (collision)*; *Average time per iteration* (measured in milliseconds); and *Number of iterations made until reaching the goal*. Below we show a summary of the results obtained.

As in the *deterministic* model, the algorithm was executed 8 thousand times with the base case setup, in order to obtain a sufficiently dense data set to evaluate the effect of the algorithm parameters.

Fig. 5.3 shows distributions of the metrics similar to the ones in ***Fig. 3.4***. For example, at least 75% of the executions in the base case obtained 0% of iterations in collision, a mean of 0.107%, and the maximum is 4.02%.

In ***Fig. 5.3.center*** (average time per iteration), quartiles Q_1, Q_2, Q_3, have values of $74.31, 77.95, 81.58$, respectively, while in the *deterministic* case they have values of $71.75, 73.25, 75.8$, respectively. The mean has a value of $77.99\ (ms)$, and in the *deterministic* case $71.77\ (ms)$. Due to the manual adjustment of extra configuration parameters, the mean time per iteration tends to be slightly higher than in the deterministic model, but generally close enough to make comparisons between both.

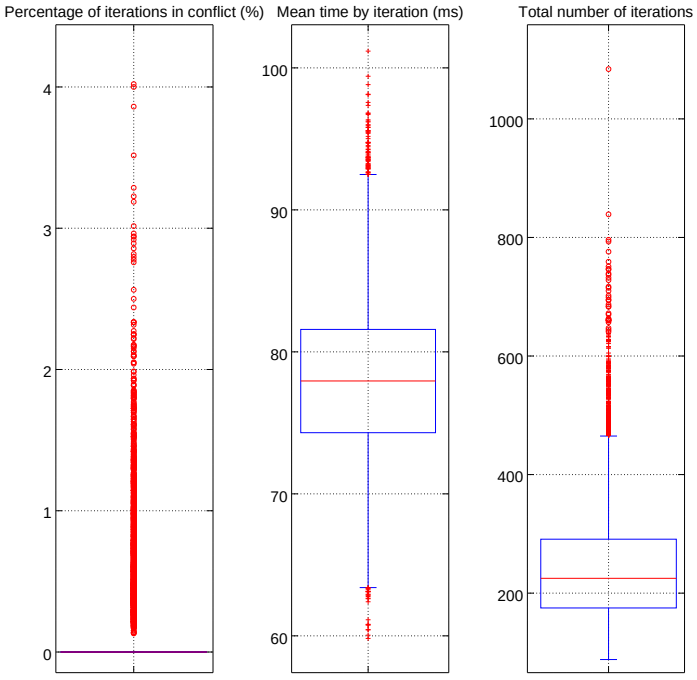


Figure 5.3: Three *boxplots* representing the distribution of the performance metrics proposed on the base case.

In ***Fig. 5.3.right*** (total number of iterations), quartiles Q_1, Q_2 (median), Q_3, have values of $175, 225, 291$, respectively, while in the *deterministic* case they show the values of

$136, 151, 174$. In the probabilistic model, the number of iterations to reach the goal is between 25% and 70% greater, with this base configuration. This is because the robot has more constraints in the velocity space, induced by the sampling of VOs. Also, not being a boolean approach, but continuous, the robot takes velocities of smaller magnitude. Below is a summary of the results obtained by varying the configuration parameters of the experiments.

Variation of the parameter nH

The number of simulated humans directly influences the density of the environment. Since its dimensions are constant $(20m * 20m)$, the density corresponds to $\frac{nH}{400}\frac{humans}{m^2}$. Recalling that the base case has a total of $50\ humans$, then its density is $0.125\frac{humans}{m^2}$. The values in which the parameter nH was varied and their respective densities are the following: 17: $0.0425\frac{humans}{m^2}$; 25: $0.0625\frac{humans}{m^2}$; 34: $0.085\frac{humans}{m^2}$; 50: $0.125\frac{humans}{m^2}$ (base case); 67: $0.1675\frac{humans}{m^2}$.

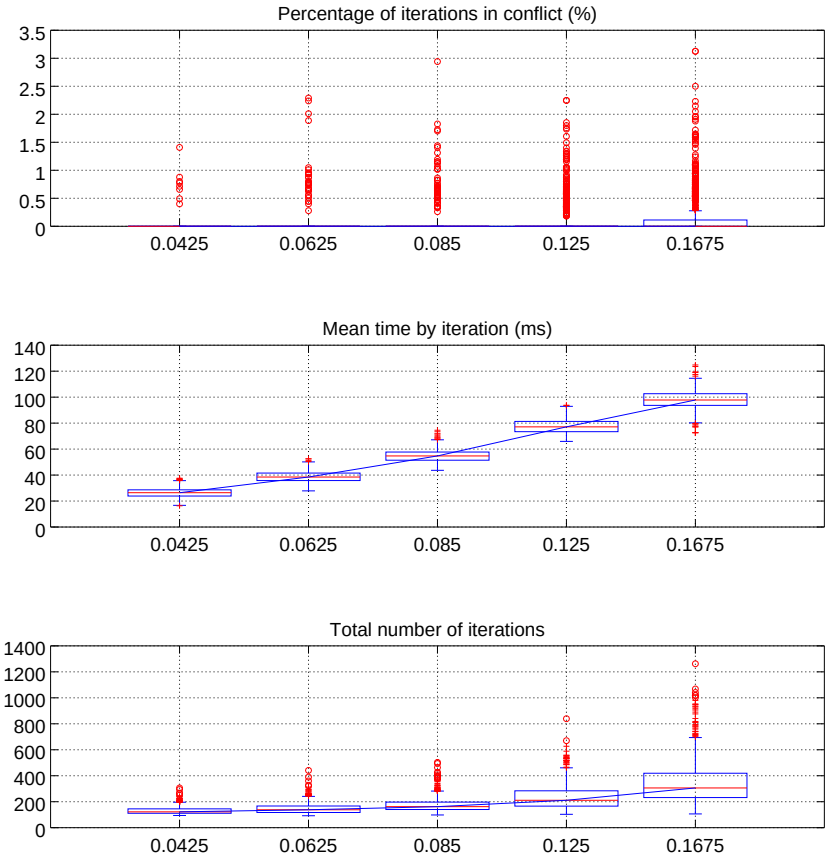


Figure 5.4: Performance metrics on the testing sets by varying the parameter nH. The x axis shows the value of the human density of the environment, measured in $\frac{humans}{m^2}$. With $0.125\frac{humans}{m^2}$ in the base case.

In ***Fig. 5.4***, we can see that in this approach, by increasing the density of predicted trajectory models (remember that each human now provides 3 models), the metrics are negatively affected, as in the deterministic approach. But in this case, the increase in density affects less the metrics. ***Fig. 5.4 top*** shows that the percentage of iterations in conflict is negatively affected, as expected, by increasing the density of the environment.

The average time per iteration is also affected by increasing the density, as in the deterministic model. By generating more velocity obstacles, the calculation of the optimal velocity is slowed down, since the filling of the grid is carried out with more operations. Coupled with this, the denser the environment, the more likely it is to relax the constraints in time, adding more complexity to the iterations. In ***Fig. 5.4 center***, we can see that the average time per iteration remains acceptable, i.e., in all configurations, Q_3 stays below

$\Delta t = 350\ ms$.

The number of iterations made until reaching the goal also increases as the density of humans in the environment increases. In ***Fig. 5.4 bottom***, we can see that in experiments with lower density, the number of iterations to reach the goal is relatively small, with outliers smaller than 350.

Variation of the parameter nOPercentage

Fig. 5.5 shows the behavior of the three performance metrics by varying the parameter $nOPercentage$. Recall that the base case takes into account 50 humans, i.e., 150 prediction models, of which nO corresponds to 50%. The values in which the parameter $nOPercentage$ was varied are the following ones: 10%: 5 humans; 25%: 12 humans; 50%: 25 humans (base case); 80%: 40 humans; 100%: 50 humans.

Fig. 5.5 top shows that, in general, the percentage of iterations in conflict is mostly 0% when the robot *can see* at least 10% of the humans.

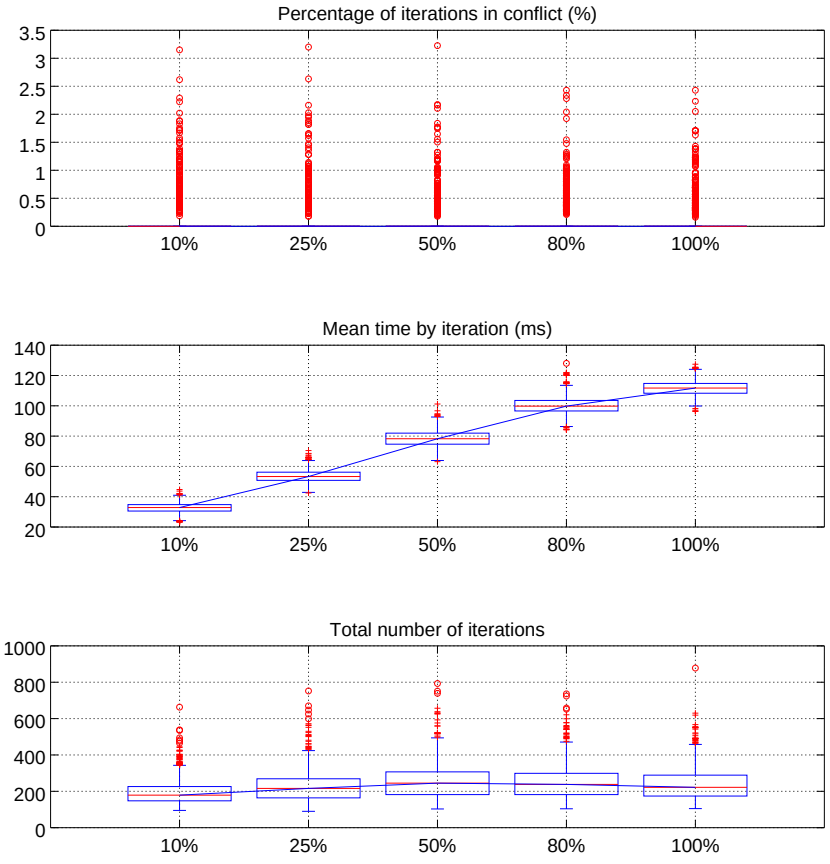


Figure 5.5: Performance metrics varying the parameter $nOPercentage$. The x axis of the three graphs shows the percentages of nO over the number of humans, (50). With 50% in the base case.

Fig. 5.5 top shows that, in general, the percentage of iterations in conflict is mostly 0% when the robot *can see* at least 10% of the humans.

In ***Fig. 5.5 center***, we observe that, when increasing the percentage of humans that the robot can see, the average time of computation per iteration increases, as expected, because a greater number of humans will produce a greater number of candidates, and in the same way, $ACV_A \backslash VO_A^\tau$ can be mostly occluded, producing more relaxation steps. The distributions when varying this parameter, both in the deterministic model and in the probabilistic one, are very similar, but the outliers are distributed closer to the median. In general, in the base case, the average time per iteration is *acceptable* (less than Δt), and in the worst case, the maximum outlier remains less than Δt.

Finally, as in the deterministic model, in ***Fig. 5.5 bottom***, we can see that the number of iterations to reach the goal is practically unaffected by varying the number of visible humans.

Variation of the parameter ComfortRadScale

The ***fig. 5.6*** shows the behavior of the three performance metrics by altering the parameter $ComfortRadScale$. Recall that, in the base case, we have used a scale of 1.1, with respect to r_A. The values in which the parameter was varied are the following: 1.0; 1.1 (base case); 1.5; 1.8; 2.0.

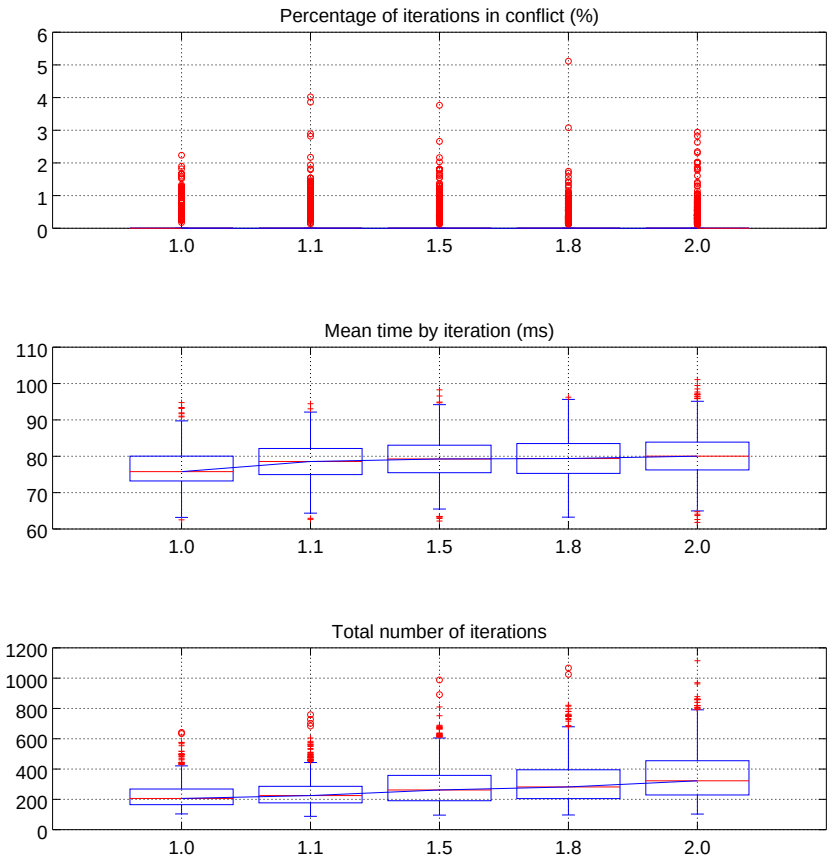


Figure 5.6: Performance metrics by varying the parameter $ComfortRadScale$. The x axis shows the scale of the comfort radius with respect to the radius of the robot. With 1.1 in the base case.

In ***Fig. 5.6 top***, we observe that the median of the percentage of iterations in conflict, for all the configurations of the comfort radius, shows a value of 0%. Intuitively, by increasing the comfort radius of the robot, its space of non-conflicting velocities is more restricted, since VOs of greater dimension are generated. But, when taking into account the uncertainty, the robot acts in a more *prudent* way, causing a smaller number of iterations with collision.

The average time per iteration, in configurations of comfort radius greater than the radius of the robot, are practically invariant and slightly greater than in the deterministic case. See ***fig. 5.6 center***.

The size of the comfort radius influences, in much the same way as for the deterministic model, the total number of iterations to reach the goal, as seen in ***Fig. 5.6 bottom***.

Variation of the parameter RestAccScale

Fig. 5.7 shows the behavior of the three performance metrics by altering the parameter $RestAccScale$. The values in which the parameter $RestAccScale$ was varied are the following: 1.0; 1.2; 1.5; 1.8 (base case); 2.0.

The median percentage of iterations in collision, using configurations of 1.0 and 1.5 for this parameter are slightly better than in the deterministic case. In ***Fig. 5.7 top*** it can be seen that, in all configurations, the median percentage of iterations in collision is equal to 0%.

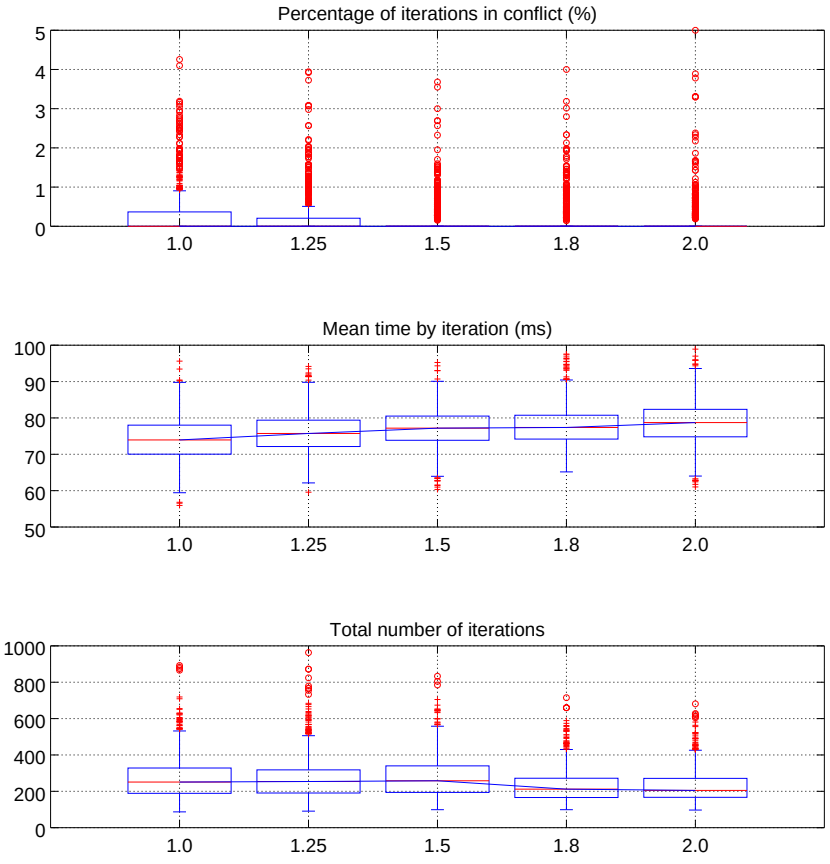


Figure 5.7: Performance metrics by varying the parameter $RestAccScale$. The x axis shows the scale of maximum accelerations for the robot with respect to its maximum control velocities. With 1.8 in the base case.

Both the average time per iteration, and the total number of iterations, are not very affected, as in the deterministic model, ***see Fig. 5.7 center, bottom***.

Variation of the parameter Segments

Higher numbers of segments (chords) in which the predicted trajectories for dynamic obstacles are divided, generate better approximations of the VOs for non-linear trajectories. Intuitively, by having more segments, we can have a better estimation for $\mathbf{v}_A^{opt}$. However, because it gradually increases the complexity of the calculation, it is convenient to use a conservative approximation that does not compromise the average time per iteration. ***Fig. 5.8*** shows the behavior of the three performance metrics by altering the parameter $Segments$. The values in which the parameter $Segments$ was varied are the following: 1

(conventional VO); 2; 3; 4; 5 (base case); 6; 7; 8; 9; 10.

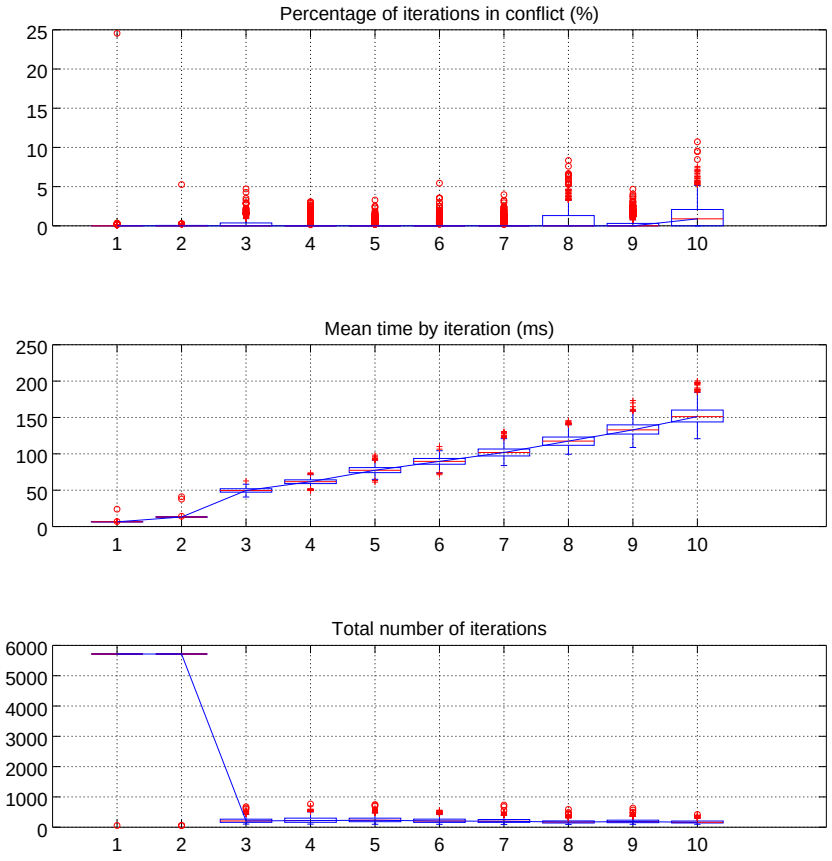


Figure 5.8: Performance metrics by varying the parameter $Segments$. The x axis shows the number of segments (chords) used for the predictions. With 5 in the base case.

The percentage of iterations in collision is not greatly affected by the number of segments in which the trajectories are discretized. The median of all configurations has the value of 0%, except in the $10 - segment$ configuration. In that case, it takes a value of around 1%, although the increase is very small. It occurs because, having a greater number of segments, the relaxation of the constraints considers, in its last step, for the base case, a time of prediction of only 0.5 seconds, i.e., the time corresponding to less than two iterations. ***See fig. 5.8 top***.

The average time per iteration is gradually affected by the number of segments. This is because the number of steps in the constraint relaxation process is equal to $Segments - 1$.

Moreover, each step increases the execution time considerably, when the number of cells is large. Unlike the deterministic model, the outliers, in the base case, never exceed, and are even much lower than, Δt. This is very important because it gives us greater certainty that the calculation of $\mathbf{v}_A^{opt}$ will be done *on time*, which, in the deterministic model, we only achieve with $Segments \leq 5$. ***See fig. 5.8 center***.

The variation of the number of segments is interesting in this case, since for $Segments = 1$ (conventional VO) and $Segments = 2$, the model does not reach its goal in most experiments, but it does reach it for 3 or more. For these configurations, the behavior is very similar to that of the deterministic model, since a tendency to increase the number of segments is not clear. ***See fig. 5.8***. The cases in which the robot does not reach the goal are caused by the duration in which a segment is traced, as well as by the process of relaxation of the constraints. In this approach, no relaxation is performed on nO, only in time, by removing the furthest segments of the predicted trajectory. In the case of conventional VO, the restrictions are never relaxed, and for the case of 2 segments, the duration of the predictions, after relaxation, is 2.5 seconds. In both cases, it is not enough to find a velocity that brings the robot closer to its goal, so it prefers to take a *safe* velocity.

Variation of the parameter τ

Fig. 5.9 shows the behavior of the three performance metrics when altering the parameter τ. The values in which the parameter τ was varied are the following: $2.5\ s$; $3.5\ s$; $5.0\ s$ (base case); $7.5\ s$; $10\ s$.

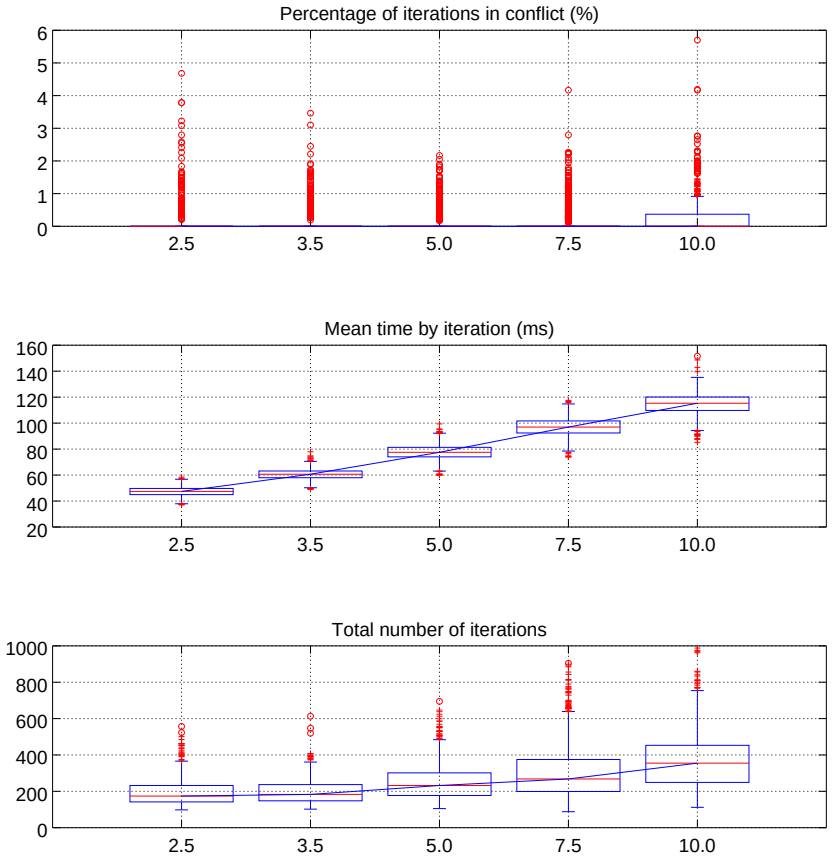


Figure 5.9: Performance metrics by varying the parameter τ. The x axis shows the duration in seconds of predictions. With $5.0\ s$ in the base case.

The percentage of iterations in collision, for the different configurations of τ, shows a mean of 0%, i.e., it is not a crucial parameter for this metric, as seen in. ***Fig. 5.9 top***.

Unlike the deterministic model, a tendency in the average time per iteration is clearly visible when the parameter τ varies. The larger it is, the slower the calculation of $\mathbf{v}_A^{opt}$. Although the average time per iteration is greater with this model, the outliers remain below the value of Δt, contrary to the first. Also, the total number of iterations increases as the value of τ increases, ***see fig. 5.9 center, bottom***.

Variation of the parameter Δt

Fig. 5.10 shows the behavior of the three performance metrics by altering the parameter Δt. The values in which the parameter Δt was varied are the following: $0.1\ s$; $0.25\ s$; $0.35\ s$ (base case); $0.5\ s$; $0.75\ s$.

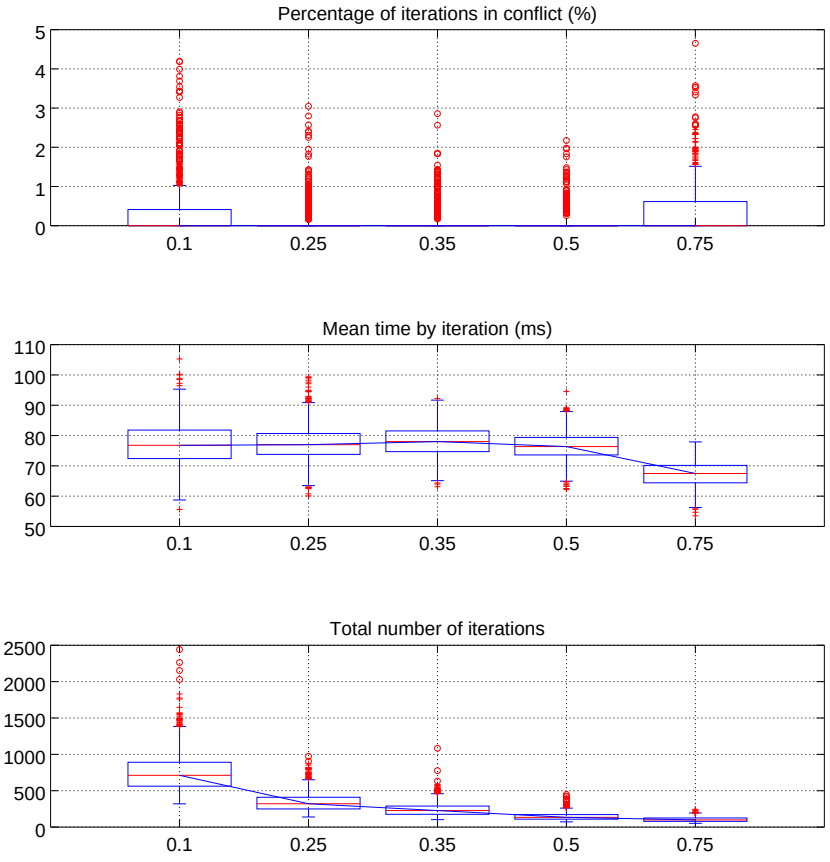


Figure 5.10: Performance metrics by varying the parameter Δt. The x axis shows the duration in seconds of this parameter. With $0.35\ s$ in the base case.

Fig. 5.10 top shows that, when varying Δt, the median percentage of iterations in conflict is not affected, taking the value of 0% in all configurations.

In all cases of variation of this parameter, in the deterministic model, the maximum outliers are always below the value of Δt, that is, the average time per iteration is always acceptable. Now, for the probabilistic model, the distributions are very similar, but the

outliers remain relatively close to the medians, contrary to the deterministic case. ***See fig. 5.10 center***.

Finally, in ***Fig. 5.10 bottom***, one can see that smaller ACVs, caused by lower Δts, cause a higher number of total iterations. The distributions of both the deterministic and the probabilistic models are very similar, even in their outliers, when varying Δt.

Variation of the parameters columns and rows

In all the configurations for the experiments, we use the same value for the number of rows and for the columns. The values in which those parameters were varied, and the number of cells generated, are the following: $20; 400$; $50; 2500$; $100; 10000$ (base case); $140; 19600$; $200; 40000$.

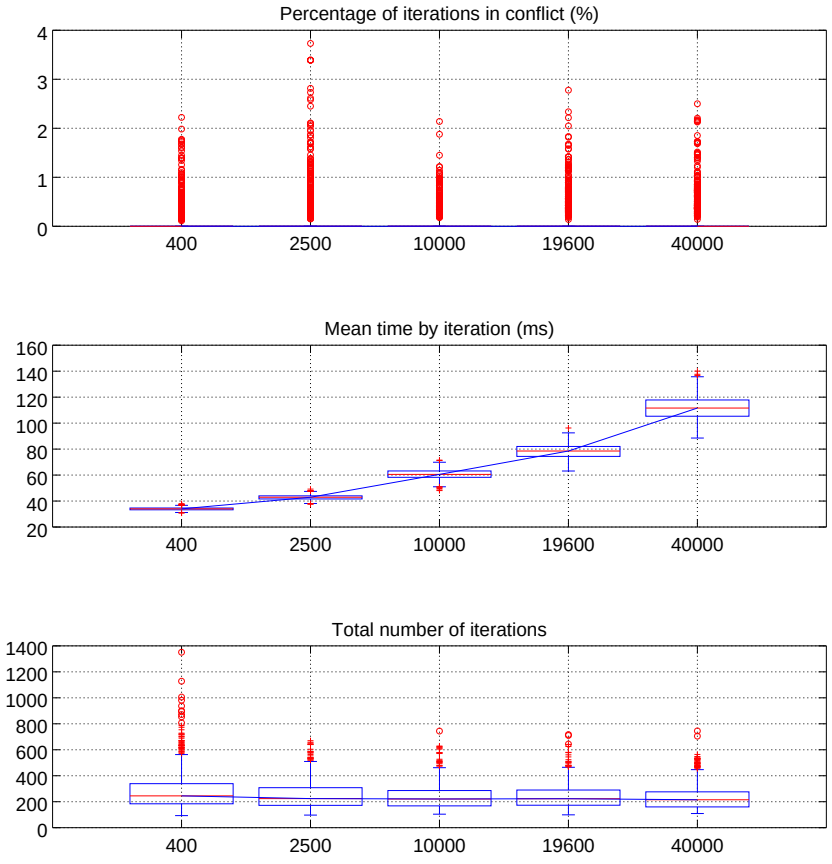


Figure 5.11: Performance metrics by varying the parameters $columns$ and $rows$. The x axis shows the number of cells in the grid. With 10000 in the base case.

The percentage of iterations in collision, and the total number of iterations, is not affected by the fineness of the grid, using these values. The first of these metrics shows, in all configurations, medians equal to 0%. Meanwhile, the number of iterations, also in all configurations, stays roughly around a single value, which is around 230 iterations. ***See fig. 5.11 top, bottom***.

The average time per iteration is affected by increasing the number of cells. Because the dimensions of the ACV are invariable to these parameters, when the discretization is so fine, adjacent cells represent very similar velocities, and this precision may not be significant. ***See fig. 5.11 center***.

Variation of the parameter samples

This parameter allows us to handle the uncertainty of the distribution of the prediction models. The more samples we draw, the more accurate the distribution estimate will be. As the sampling process is the most computationally intensive part of the algorithm, greater precision means also increasing the average time per iteration. The values in which the parameter $samples$ was varied are the following: 50; 75; 100 (base case); 125; 150.

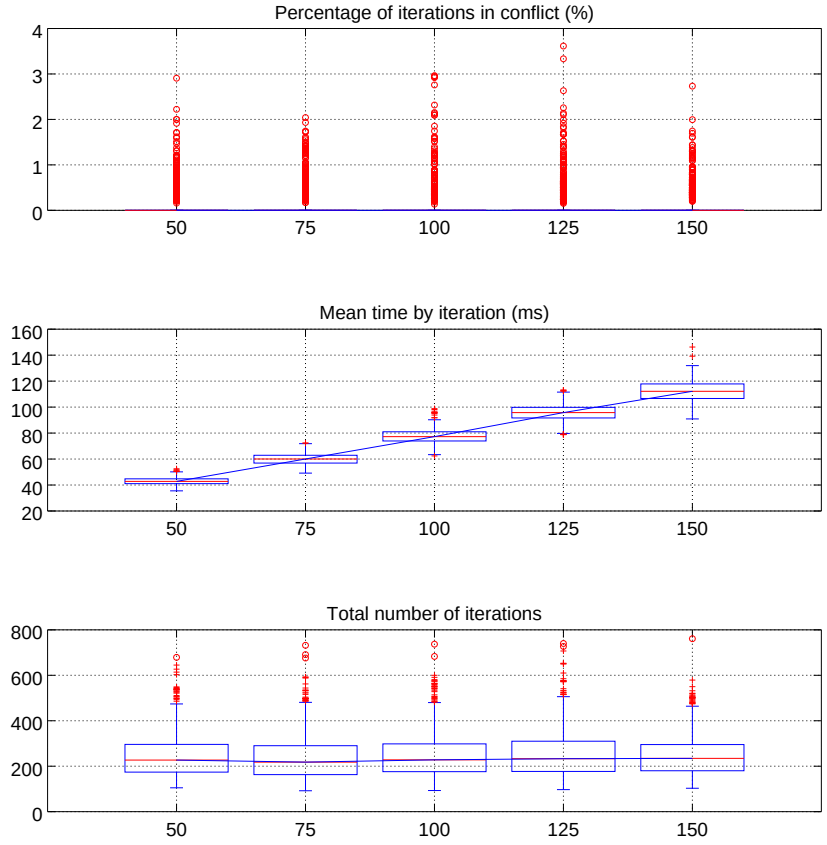


Figure 5.12: Performance metrics by varying the parameter $samples$. The x axis shows the number of $samples$. With 100 in the base case.

The number of samples used in the experiments does not affect the median of the distributions of the results obtained for the first metric. As when varying other parameters,

the medians take the value of 0% of iterations in conflict. ***See Fig. 5.12 top***.

Fig. 5.12 center shows a linear trend between the average time per iteration, and the number of samples, which is logical, since the number of generated velocity obstacles per iteration is $samples * segments * nO$.

When varying the number of $samples$, no significant variation in the total number of iterations is seen. The means of the results obtained for each configuration are roughly the same, around 210 iterations, as shown in ***fig. 5.12 bottom***.

Variation of the parameter threshold

With the $threshold$ parameter, we choose how risky the robot decisions should be, i.e., the maximum collision probability that we allow the robot to consider when choosing $\mathbf{v}_A^{opt}$. The values in which the parameter $threshold$ was varied are the following: 0.2; 0.2 (base case); 0.3; 0.4; 0.5.

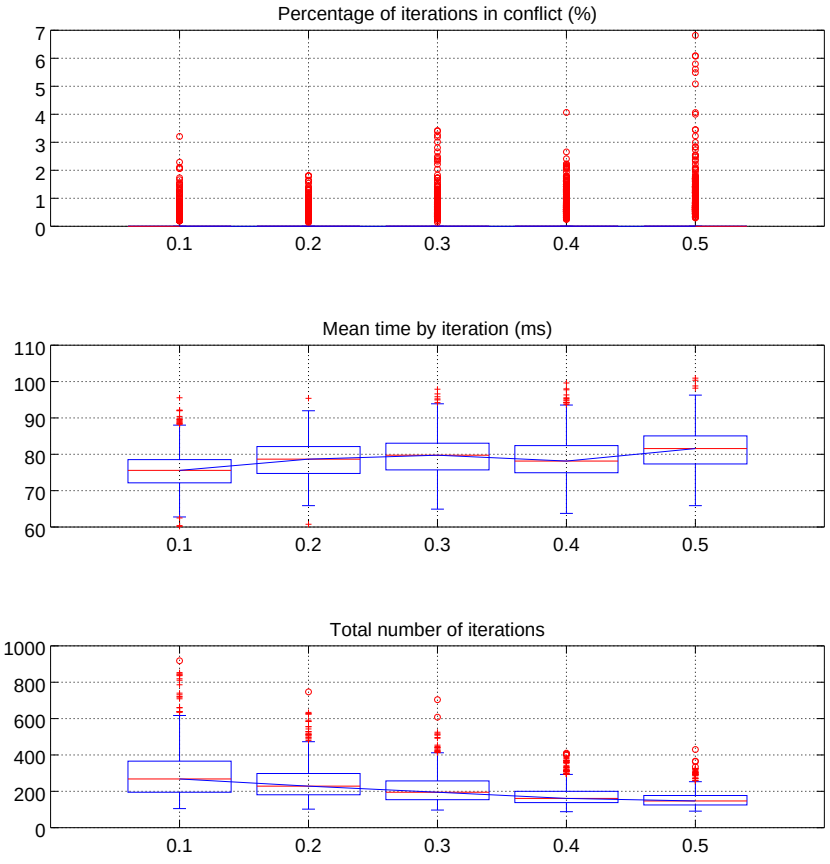


Figure 5.13: Performance metrics by varying the parameter $threshold$. The x axis shows the value of this parameter. With 0.2 in the base case.

The percentage of iterations in collision in all the configurations of this parameter results in similar distributions. All of these show medians with 0% of iterations in collision. Again, this metric is not affected by the variation of the parameters on the base case. ***See fig. 5.13 top***.

The average time per iteration has no clear trend when the $threshold$ is varied, ***See fig. 5.13 center***. However, there is a slight increase in the median of the data sets of the results. Although in theory this metric should decrease if the $threshold$ is increased, the difference between the means of when $threshold$ takes the values of 0.1 and 0.5, is approximately $5\ ms$, a very small value to be considered significant.

The total number of iterations does clearly improve when larger $threshold$ values are used, ***see fig. 5.13 bottom***. This is because the robot acts in a less *prudent* way when

choosing $\mathbf{v}_A^{opt}$, i.e., the search process is more permissive and finds the optimum faster.

Variation of the parameter timeToCollision

The *time to collision*, in combination with the magnitude of the maximum achievable velocity by the robot, is the parameter we use to quickly and lightly (but imprecisely), discard the VOs that might not have an intersection with the ACV. The higher this parameter, the greater the test acceptance radius. If we configure this parameter with a very low value, many of the VOs could be discarded incorrectly, causing collisions. The values in which the parameter $timeToCollision$ was varied are the following: 0.5; 1.5; 2.5 (base case); 3.5; 5.0.

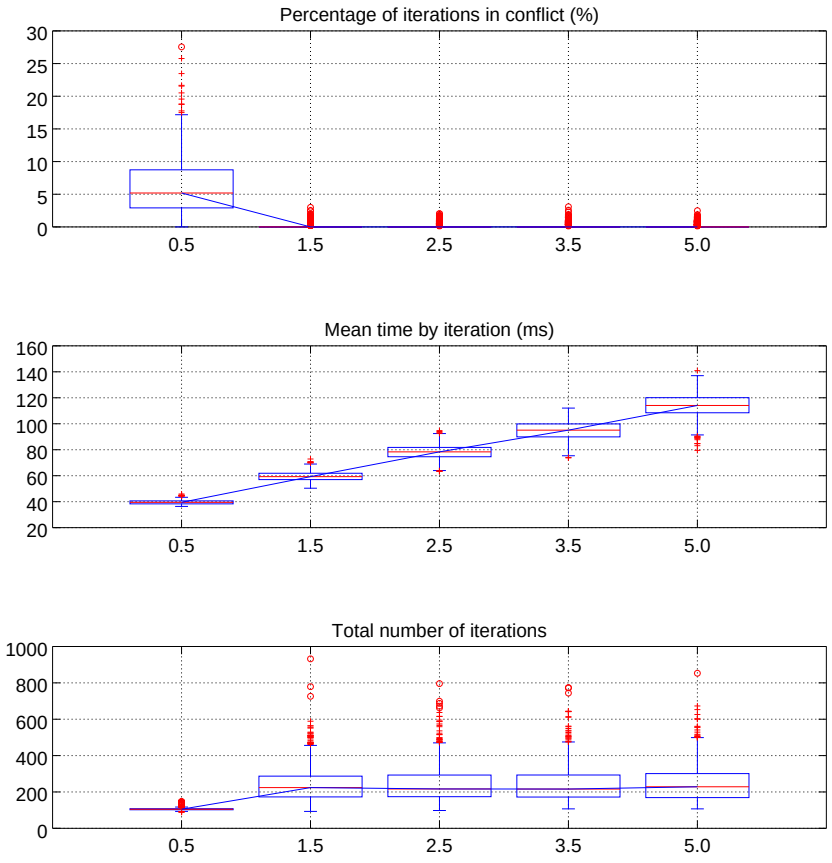


Figure 5.14: Performance metrics by varying the parameter $timeToCollision$. The x axis shows the duration in seconds of this parameter. With 2.5 s in the base case.

In ***fig. 5.14 top***, we can see that, with a value of 0.5 seconds in this parameter, it results in a distribution with a median of approximately 5% of iterations in conflict. That is, this parameter affects the first metric to some extent, since the other configurations of it show very similar distributions, all with a median of 0% of iterations in collision.

The average time per iteration shows a linear relationship with the time to collision chosen. This is because, by increasing the acceptance radius of the test, the number of velocity obstacles that will be processed increases. The generated velocity obstacles increase their dimensions as they move away from the robot, i.e., they represent velocities of greater magnitude, and their centroids move away with the same proportion. Assuming that mobile obstacles are evenly distributed, the relationship tends to be linear. ***See fig. 5.14 top***.

The ***fig. 5.14 bottom*** shows an interesting behavior according to the percentage of iterations in conflict. When using a value of 0.5, the number of iterations is relatively low, compared with the other configurations, which present very similar distributions, with a median of approximately 210 total iterations.

Some additional experiments, with a new base case

In the base case configuration described above, to compare both models, we have used values that, in our opinion, are very high and do not provide a considerable improvement, due to the iterative calculation of the optimal velocity. Hence, we decided to execute some additional experiments, varying these parameters, as well as the $threshold$, in order to achieve an improvement in the execution time, and to determine whether a smaller number of cells and samples could still not compromise the evasion of obstacles.

1000 executions of the algorithm were performed with each of these configurations. For the base case that we consider now, we use the same values of most parameters as in the already discussed experiments, and for the parameters that we vary, we consider the following configuration:

$threshold$: The maximum collision probability value that we are willing to accept when choosing the optimal velocity, *(0.005)*.

$samples$: The number of samples to estimate the probability of collision induced by a Velocity Obstacle, *(20)*.

$rows$, $columns$: The dimensions of the grid. *(*20 *and* 20*, resulting on* 400 *cells)*.

Variation of the parameter threshold

The values in which the parameter $threshold$ was varied are the following: 0.005 (base case); 0.01; 0.05; 0.1; 0.15.

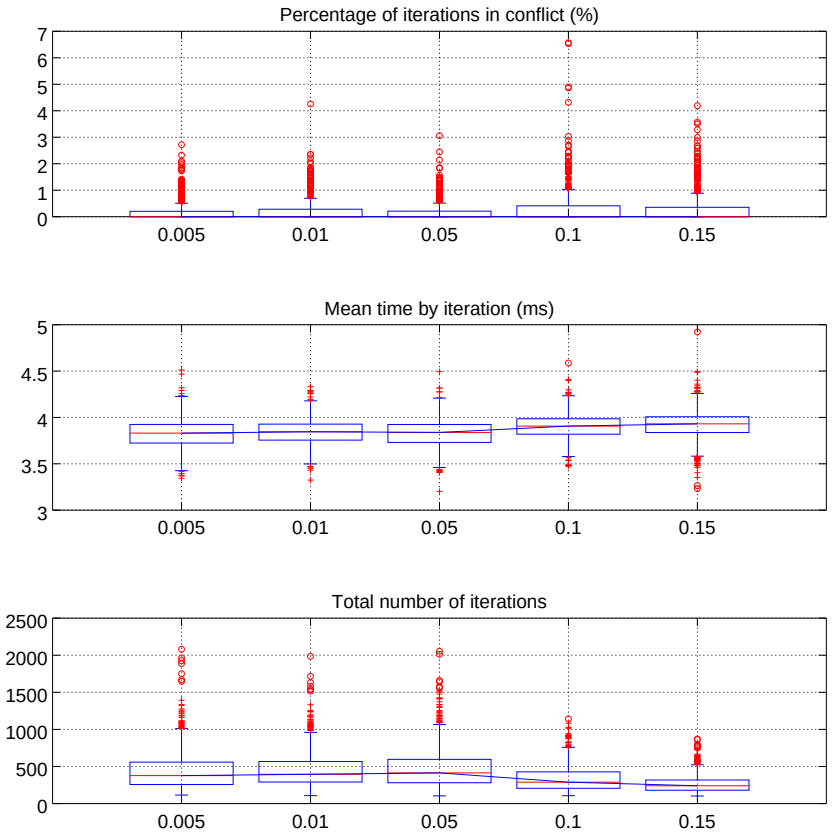


Figure 5.15: Performance metrics by varying the parameter $threshold$. The x axis shows the value of this parameter. With 0.005 in the base case.

The variation of $threshold$, using this new base case, shows a slight improvement in the distribution of the results. Quartiles Q_3 show values close to half a percentage point, while the median remains at 0%. The most important metric in these comparisons is the average time per iteration, which presents a considerable improvement in the calculation of $\mathbf{v}_A^{opt}$, with respect to the original base case. The original base case showed medians around $80\ ms$ of time per iteration and, for this new configuration, it is around $4\ ms$. This

represents an improvement of $20x$ the computation speed.

The third metric shows higher values in the medians, compared to the original base case, with approximately twice the total number of iterations until reaching the goal.

Variation of the parameter samples

The values in which the parameter $samples$ was varied are the following: 10; 20 (base case); 30; 40; 50.

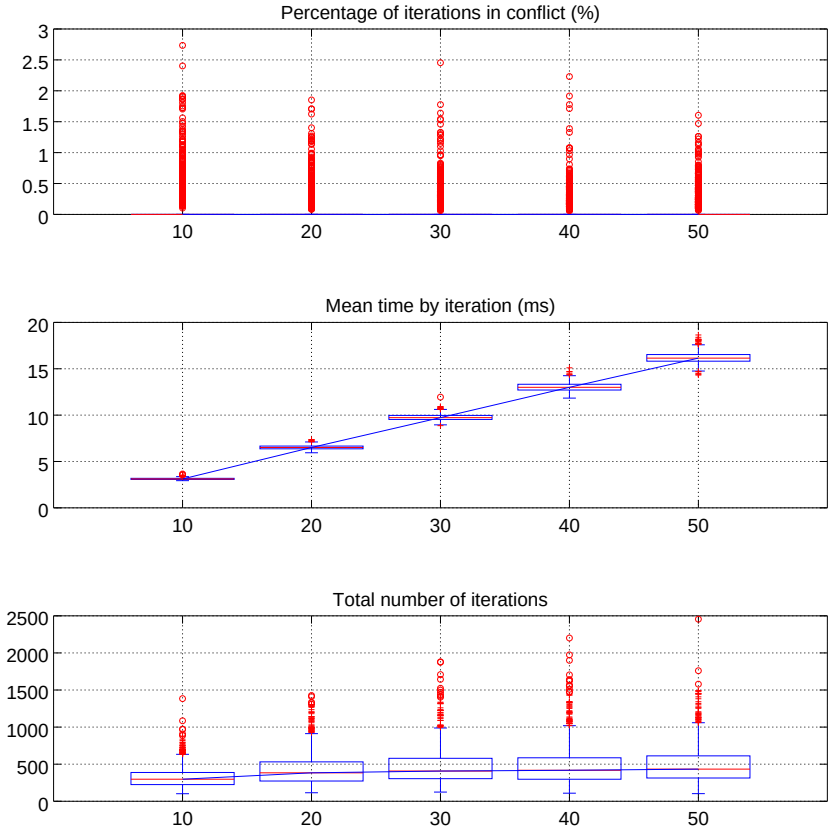


Figure 5.16: Performance metrics by varying the parameter $samples$. The x axis shows the number of $samples$. With 100 in the base case.

The performance shown by varying this parameter are very similar to the original base case. The medians in the first metric show values of 0% iterations in collision. In both base

cases, an apparently linear relationship can be seen between the number of samples and the average time per iteration.

The total number of iterations to reach the goal presents medians around 400, which is approximately double the one of the original base case.

Variation of the parameters columns and rows

In all the configurations for these experiments, we use the same value for the number of rows and for the columns. The values in which those parameters were varied, and the number of cells generated, are the following: $10(100)$; $20(400)$ (base case); $30(900)$; $40(1600)$; $50(2500)$.

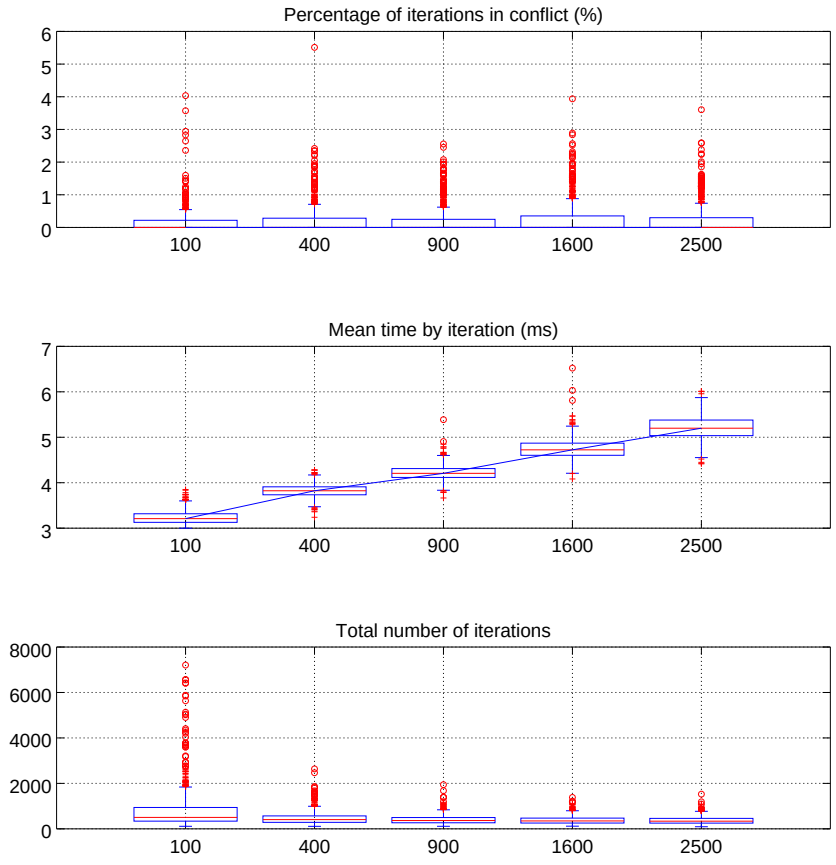


Figure 5.17: Performance metrics by varying the parameters $columns$ and $rows$. The x axis shows the number of cells in the grid. With 1000 in the base case.

In the same way that varying the other mentioned parameters, the medians of the resulting data take values of 0% of iterations in collision. The average time per iteration, and the total number of iterations show distributions very similar to the original base case, the former being approximately $20x$ faster than in the original base case, and the latter showing values approximately equivalent to twice the results with the original case.

Chapter 6

Conclusions

In this thesis, the problem of robot navigation in crowds has been addressed by using an approach based on the well known concept ot velocity obstacles. The modeling of velocity obstacles using what we call Quadrilateral Velocity Ostacles ($QVOs$) gives us a conservative approach that does not compromise the behavior of the robot to avoid collisions, because although it unnecessarily restricts some portions of the velocity space, this is not critical because these portions are small in size. As it is represented by a polygon of four points in the velocity space, it is very easy and quick to evaluate whether a candidate for optimal velocity is inside or outside the set of restricted velocities.

The assumed prior knowledge of the trajectory that a mobile obstacle will follow, within a time span τ, allows the robot to foresee a better strategy to avoid collisions with that mobile obstacle. Two approaches dealing with the navigation problem have been proposed, using a deterministic fashion and a probabilistic fashion. In the deterministic model, it is assumed that the trajectory of the mobile obstacles is really traced as predicted. In the real world, of course, this is not the case, because mobile obstacles, be they human or other robots, can change their trajectory unexpectedly. With the deterministic model that we have proposed, it depends on the capacity of reaction of the robot to properly handle these situations.

In general, the performance of the deterministic model shows good results, since in most of the experiments, the median percentage of iterations in collision resulted in 0%, although some outliers were up to 20%, for the more difficult configurations of the execution parameters.

The density of obstacles (simulated humans) in the environment greatly affects the average time by iteration. In the experiments configured with a density of $0.5\ \frac{humans}{m^2}$, or when nO, the number of obstacles that the robot *can see*, is high, i.e., greater than 80% of the number of humans, the value of the median of the average time by iteration results just at an acceptable value, i.e., less than or equal to Δt. It presents, sometimes, outliers

that are too large and that could slow down the decision making of the robot. This is one of the most important aspects, because the robot must always be able to calculate an optimal speed within the duration of the control sampling rate Δt. Despite these limitations, the deterministic model results in a good approach to solve the problem of collision avoidance among crowds. It is important to mention that, in the execution of the experiments, the behavior of the robot does not show oscillations like the ones shown using the conventional VO approach and RVO [9].

The probabilistic model does not present oscillations, since it represents non-binary values for the choice of $\mathbf{v}_A^{opt}$, because the probability of collision distribution shows a smoothed function. This approach presents improvements in the metric of average percentage of iterations in collision, in comparison with the deterministic model. This is because with this approach, the robot has the ability to act while considering the uncertainty in the trajectory predictions of the mobile obstacles.

When looking for the optimal velocity over a simplified representation of the entire search space, using a grid in the set of attainable cartesian velocities ACV (dynamically feasible velocities), it is possible to choose velocities not only depending on the preferred one, but also, if necessary, to prioritize choosing velocities that keep the robot away from probable collisions.

As previously mentioned, most of the parameters of the probabilistic approach were adjusted to those of the deterministic model, with $\frac{1}{3}$ of the humans used in that model, resulting in the same number of predicted trajectories. This is to achieve a similar average time by iteration in both approaches, which allows fairer comparisons in the base case. After the comparisons, we can observe that the distributions in this metric are practically the same, differing only in the dimension of the outliers.

One might think that it is not very important to consider the outliers, but sometimes, in the deterministic case, they show unacceptable values. For example, when varying the number of segments, we can observe outliers of up to $4\ sec$ of magnitude, when in this case $\Delta t = 0.35\ seg$. In these cases, it is likely that an iteration will not be calculated within the duration of Δt, which would cause a chain reaction of delays in computation time. This is because when applying the calculated velocity, the environment would not correspond anymore to the one observed before the calculation. Contrary to this, the probabilistic model does not show outliers with magnitude greater than Δt, i.e., the calculation of the optimal velocity is always made on time.

The greater the number of cells in the grid, the greater the precision of the choice of the optimum velocity, and likewise, the estimation of the distribution of the VOs will be more accurate with a greater number of samples.

With our second base case, the combination of the parameters $threshold$, $samples$, $rows$ and $columns$, results in a slightly higher percentage of collision iterations, although the median distributions remain at 0% and the value of the quartiles Q_3 is slightly greater than 0%. The greatest impact caused by these configurations is the average time per iteration, showing in most cases, a speed-up of $20x$. In contrast, the total number of iterations is, in general, approximately twice the number obtained with the original base case.

A recommended configuration should make a balance between the three metrics, firstly minimizing the percentage of iterations in conflict, then ensuring that the average time per iteration is less than Δt (in all the results this is fulfilled), and third, minimizing the total number of iterations to reach the goal. Based on the data obtained, a proposal could be to use $threshold$ between 0.005 and 0.05, to minimize the first metric; $rows$ and $columns$ with values between 20 and 40, to minimize the number of iterations; finally a value of 50 in $samples$, also to minimize the first metric.

6.1 Future work

The work to be done in the future consists of implementing the algorithm in experiments in the real world. So far, some experiments have been carried out with the help of the Simulator class using OpenFrameworks. Experiments were also carried out with simulated Turtlebot robots in Gazebo [25], integrating our library with a control based on ROS.

In order to carry out experiments in a real environment, an ***OptiTrack*** motion capture system will be used to measure the robots' poses in real time and the position of other obstacles within the environment, through an array of 13 cameras that capture the positions of markers mounted on the robots and obstacles. ***See Fig 6.1***

The ROS (*Robot Operating System*) framework is used to communicate the calculated velocities to the robot and to tell the robot perform them. ***Fig. 6.2*** shows the communication diagram. The communication node receives the poses of the mobile agents/obstacles as well as the pose of the robot. Then, these data are processed and sent to the *Prediction Node*, where they are processed, and as a result, we obtain predicted trajectories in a time horizon, which are received by the node responsible for calculating the optimal velocity. Then this velocity is transmitted to the communicator node, which is responsible for transforming it into control inputs according to the dynamics of the robot (no holonomic in this case), and finally, the result is sent to the robot for its execution.

It is also intended that, as future work, we implement a version of both approaches, the determinist and the probabilistic, in $3d$ environments. The main idea would be the same: use simple polyhedra to inscribe the VOs, more specifically hexahedra, ***see fig. 6.3***.

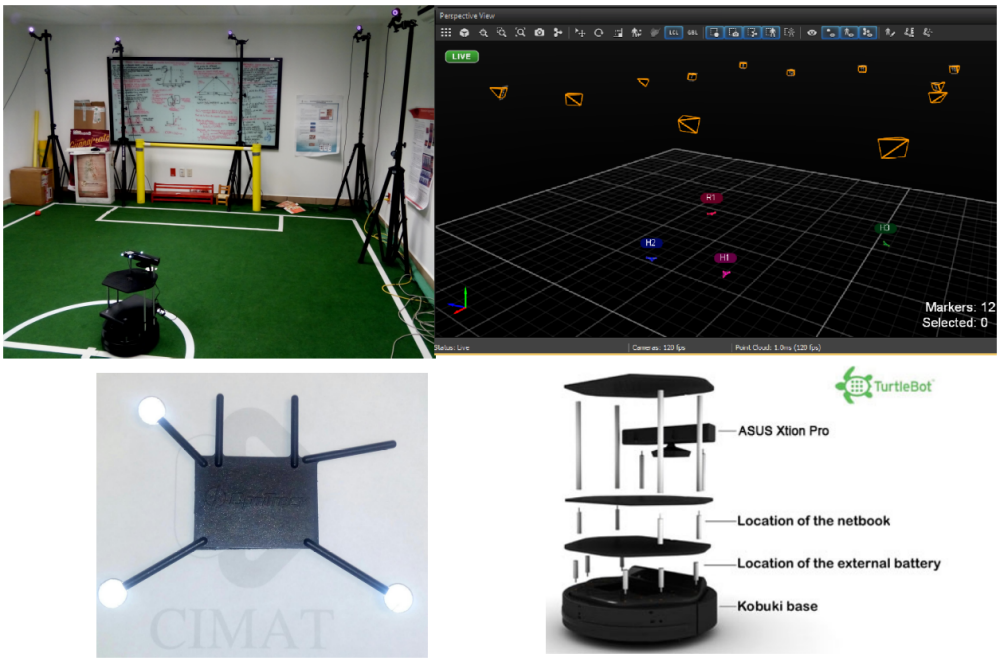


Figure 6.1: The first image shows the real environment of an experiment and one of the robots. Also, five cameras of the motion capture system are observed. The second image shows the virtualization of the environment. At the top, we can see the arrangement of 13 cameras, and below four markers: R_1 corresponds to a robot and H_i to each human simulated in this experiment. The third and fourth images shows one of the markers that are mounted on each moving obstacle and more detailed diagram of ***TurtleBot 2***.

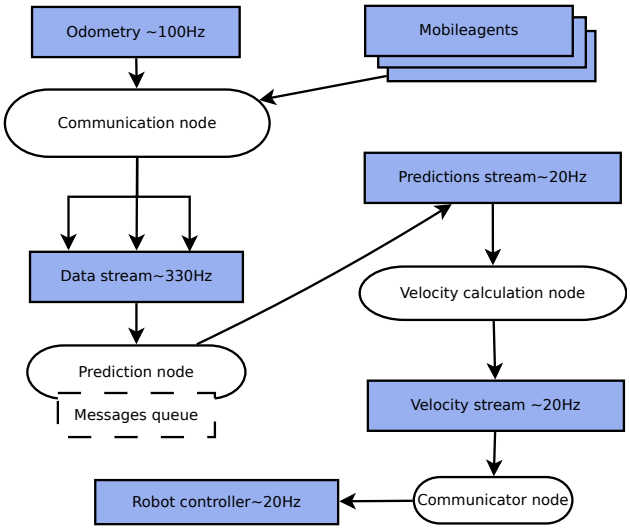


Figure 6.2: Diagram of communication between motion capture, velocity calculation and its application on the robot.

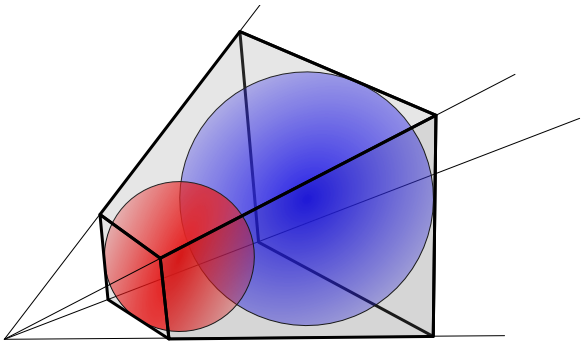


Figure 6.3: Modeling a VO in 3d using a hexahedron, bounded by two times, the red sphere represents the upper bound, and the blue sphere the lower bound.

In the same way, we would draw samples for the probabilistic model and intersect the hexahedrons with the ACV, now discretized in cuboids.

Appendix A

Trajectory Generator

To carry out the experiments, we use synthetic datasets created from a trajectory generator. Mobile obstacle trajectories are generated that do not interact with each other nor have defined goals, but instead trace random routes. These trajectories are used as input data and are complemented with simulated robots with the *Simulator class*, which interact with each other using ***Algorithms 1 and 2*** depending on the case.

To generate the trajectory of each simulated mobile obstacle, the environment must first be set, i.e., the parameters with which the experiment will be carried out must be established, being the following:

$simT$: The duration of the simulation.

$min\Delta t$: The minimum expected duration between iterations.

$max\Delta t$: The maximum expected duration between iterations.

nO: The number of mobile obstacles that are to be simulated.

$minX$: The lower limit of the x coordinate of the environment.

$maxX$: The upperr limit of the x coordinate of the environment.

$minX$: The lower limit of the y coordinate of the environment.

$maxY$: The upper limit of the y coordinate of the environment.

Given the aforementioned parameters, we can deduce Δt, the average duration between iterations. To make the generation of data a bit more realistic, we do not set $\Delta t(t_i) = \Delta t$, but in each iteration this value is obtained randomly with a uniform distribution, bounded between $min\Delta t$ and $max\Delta t$, we try to simulate a reading of observed data from a *real environment*, which in theory would be obtained with a frequency of Δt,

but in practice the duration between iterations fluctuates.

Once the environment has been configured, the nO mobile obstacles are set. Each of these has the following parameters:

$min\tau_c$: The minimum expected duration between each change of trajectory.

$max\tau_c$: The maximum expected duration between each change of trajectory.

ω: The initial angular velocity.

$min\omega$: The minimum angular velocity physically achievable.

$max\omega$: The maximum angular velocity physically achievable.

$min\dot{\omega}$: The minimum angular acceleraion physically achievable.

$max\dot{\omega}$: The maximum angular acceleration physically achievable.

ℓ: The initial linear velocity.

$min\ell$: The minimum linear velocity physically achievable.

$max\ell$: The maximum linear velocity physically achievable.

$min\dot{\ell}$: The minimum linear acceleraion physically achievable.

$max\dot{\ell}$: The maximum linear acceleraion physically achievable.

ID: The identifier of the mobile obstacle.

To make the trajectories generated more realistic, τ_c is established randomly with a uniform distribution bounded between $min\tau_c$ and $max\tau_c$, this value corresponds to the length of time in which the mobile obstacle will maintain its $\omega(t)$ and $\ell(t)$ speeds, before it changes its direction. Once that time has elapsed, $\omega(t)$ and $\ell(t)$ are altered, and τ_c is calculated again.

The new speeds are then established as: $\omega(t_{i+1}) = \omega(t_i) + \Delta\omega(t_i)$ and $\ell(t_{i+1}) = \ell(t_i) + \Delta\ell(t_i)$. Where $\Delta\omega(t_i) = \dot{\omega}(t_i) * \Delta t(t_i)$ and $\Delta\ell(t_i) = \dot{\ell}(t_i) * \Delta t(t_i)$. Similarly, the accelerations are obtained at random from uniform distributions bounded between the maximum and minimum accelerations that are physically achievable by mobile obstacle, i.e., $min\dot{\omega} \leq \dot{\omega}(t_i) \leq max\dot{\omega}$ and $min\dot{\ell} \leq \dot{\ell}(t_i) \leq max\dot{\ell}$. Both speeds are limited between their extreme values, i.e., $min\omega \leq \omega(t_{i+1}) \leq max\omega$ and $min\ell \leq \ell(t_{i+1}) \leq max\ell$.

As a result, the trajectory generator throws a data stream with plain text format, which contains all the positions and timestamps that every mobile obstacle has throughout the simulation, labeled by its ID.

Bibliography

[1] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Springer Science & Business Media, 2008.

[2] B. Krogh and C. Thorpe, "Integrated path planning and dynamic steering control for autonomous vehicles," in *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, vol. 3. IEEE, 1986, pp. 1664–1669.

[3] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," *IEEE transactions on robotics and automation*, vol. 7, no. 3, pp. 278–288, 1991.

[4] J. Minguez, "The obstacle-restriction method for robot obstacle avoidance in difficult environments," in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*. IEEE, 2005, pp. 2284–2290.

[5] J. Minguez and L. Montano, "Nearness diagram (nd) navigation: collision avoidance in troublesome scenarios," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 1, pp. 45–59, 2004.

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

[7] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using the relative velocity paradigm," in *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*. IEEE, 1993, pp. 560–565.

[8] J. Van Den Berg, S. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," *Robotics research*, pp. 3–19, 2011.

[9] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE, 2008, pp. 1928–1935.

[10] J. Snape, J. Van Den Berg, S. J. Guy, and D. Manocha, "Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles," in *Intelligent Robots*

and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on. IEEE, 2009, pp. 5917–5922.

[11] A. Levy, C. Keitel, S. Engel, and J. McLurkin, "The extended velocity obstacle and applying orca in the real world," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 16–22.

[12] S. J. Guy, J. Chhugani, C. Kim, N. Satish, M. Lin, D. Manocha, and P. Dubey, "Clearpath: highly parallel collision avoidance for multi-agent simulation," in *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM, 2009, pp. 177–187.

[13] Z. Shiller, O. Gal, T. Fraichard *et al.*, "The nonlinear velocity obstacle revisited: The optimal time horizon."

[14] J. Snape, J. Van den Berg, S. J. Guy, and D. Manocha, "The hybrid reciprocal velocity obstacle," *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 696–706, 2011.

[15] Z. Shiller, F. Large, and S. Sekhavat, "Motion planning in dynamic environments: Obstacles moving along arbitrary trajectories," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 4. IEEE, 2001, pp. 3716–3721.

[16] B. Kluge and E. Prassler, *Recursive Agent Modeling with Probabilistic Velocity Obstacles for Mobile Robot Navigation Among Humans*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 121–134. [Online]. Available: https://doi.org/10.1007/978-3-540-73422-2_6

[17] C. Fulgenzi, A. Spalanzani, and C. Laugier, "Dynamic obstacle avoidance in uncertain environment combining pvos and occupancy grid," in *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007, pp. 1610–1616.

[18] A. Lambert, D. Gruyer, G. Saint Pierre, and A. N. Ndjeng, "Collision probability assessment for speed control," in *Intelligent Transportation Systems, 2008. ITSC 2008. 11th International IEEE Conference on*. IEEE, 2008, pp. 1043–1048.

[19] E. Haines, "Point in polygon strategies," *Graphics gems IV*, vol. 994, pp. 24–26, 1994.

[20] "Boost geometry," accessed: 2017-10-23. [Online]. Available: http://www.boost.org/doc/libs/1_65_1/libs/geometry/doc/html/index.html

[21] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '84. New York, NY, USA: ACM, 1984, pp. 47–57. [Online]. Available: http://doi.acm.org/10.1145/602259.602266

[22] D. Perevalov and I. S. Tatarnikov, *openFrameworks Essentials*. Packt Publishing Ltd, 2015.

[23] K. Team. Kobuki user guide. [Online]. Available: http://kobuki.yujinrobot.com/wiki/online-user-guide/

[24] S. Halim and F. Halim, *Competitive Programming: The New Lower Bound of Programming Contests*. Lulu.com, 2013, no. v. 3.

[25] W. gazebo_ros_pkgs. Gazebo simulator. [Online]. Available: http://wiki.ros.org/gazebo_ros_pkgs/