Thesis report

Luca Marseglia

December 2024

Cover page

Contents

1 Motivation

Fusion technology offers the promise of a clean and sustainable energy supply, and fusion facilities serve as critical test environments for advancing this technology. However, the operation of these facilities involves significant logistical challenges, particularly in the transportation of heavy loads within confined spaces. Efficient and safe movement of materials and equipment is essential for maintaining smooth operations. Traditionally, single-pallet transporters have been used for this purpose, but they present limitations, including high space occupancy and costly maintenance [1].

To address these inefficiencies, cooperative ground transporters, or trolleys, have emerged as a more flexible and adaptable solution. These trolleys reduce space requirements, lower maintenance expenses, and provide increased flexibility in transporting loads of varying dimensions and payloads. Furthermore, transitioning to fleets of cooperative trolleys offers logistical advantages, such as adjusting fleet size based on operational needs and standardizing transportation systems. This standardization simplifies the management of manufacturers and supply chains, allowing for easier procurement of trolleys, spare parts, and maintenance equipment. Optimizing these transportation operations is crucial for the efficient handling of materials in fusion facilities, especially as these facilities grow more complex [1].

The integration of multi-objective optimization techniques in these systems enhances their ability to address critical constraints, such as minimizing travel time, reducing energy consumption, and ensuring collision avoidance. These techniques are particularly advantageous for optimizing ground transportation logistics in complex environments like nuclear fusion facilities.

This thesis aims to address these challenges by proposing an approach to optimize the velocity profiles of robots to prevent collisions without altering their predefined trajectories. The method involves a global controller with access to all relevant information in advance, enabling the computation of velocity profiles for multiple robots simultaneously. This approach seeks to ensure efficient coordination while avoiding the limitations of real-time decision-making. The primary objectives of this optimization are to minimize the overall task completion time and total energy consumption during operation. Additionally, the thesis explores the scalability of the proposed algorithm, ensuring its effectiveness as the number of robots and the complexity of their operations increase.

2 Background

In multi-robot systems, collision avoidance is a critical challenge due to the need for multiple robots to navigate and perform tasks in shared environments without colliding with each other. As robots operate in proximity to one another, ensuring that they can coordinate their movements efficiently and safely becomes essential for maintaining productivity and preventing failures.

Collision avoidance in multi-robot systems typically involves planning trajectories or velocities in such a way that robots do not occupy the same space at the same time. This task becomes more complex as the number of robots increases, as the system must ensure safe coordination while balancing other objectives such as minimizing travel time, optimizing energy consumption, or maximizing task efficiency.

The structure of the multi-robot system can be classified as centralized or decentralized, depending on the designer. The multi-robot system is centralized if the system has supervisory control or a central planner. If the robots make their own decisions, the system is decentralized.

2.1 Centralized Approaches

In centralized collision avoidance, a central controller manages the movements of all robots in the system. This controller has a global view of the environment and the robots' positions, allowing it to compute optimal, collision-free trajectories for all robots simultaneously. Techniques such as **Mixed Integer Linear Programming (MILP)** [2] or **Mixed Integer Non-Linear Programming (MINLP)** [3] are often used in centralized systems to formulate the problem as an optimization task, where the controller seeks to minimize an objective function (e.g., travel time or energy) while sticking to collision constraints. **Genetic Algorithms (GA)** [4] are also particularly suited for centralized approaches in optimization and path planning due to their ability to explore complex, multidimensional search spaces.

Centralized approaches benefit from their global optimality, as the entire system's configuration is considered when making decisions. This ensures that the computed solution is collision-free and often optimal for the given objectives. However, the major drawback is scalability: as the number of robots increases, the computational complexity grows exponentially, making real-time application difficult. The complexity also increases along with the complexity of the

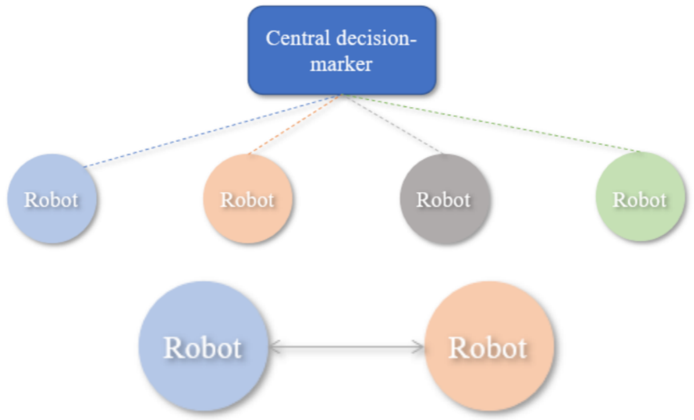


Figure 1: Framework of a centralized (top figure) and a decentralized approach (bottom figure) [5]

scenario. Centralized methods are typically used in offline planning scenarios or in small-scale systems where the number of robots is limited [5].

2.2 Decentralized Approaches

Decentralized collision avoidance methods distribute the responsibility for collision avoidance across the robots, without relying on a central controller. Each robot makes its own decisions based on local information, such as the positions and velocities of nearby robots. Algorithms like the **Reciprocal Velocity Obstacle (RVO)** are common decentralized methods. These algorithms allow each robot to predict potential collisions and adjust its velocity in real time to avoid them [6].

The advantage of decentralized approaches lies in their scalability and real-time responsiveness. Since robots compute their own collision avoidance strategies locally, decentralized systems can handle larger numbers of robots more efficiently than centralized systems. Additionally, decentralized methods are well-suited for dynamic environments, where robots must adapt to changes in the environment or the behavior of other robots. However, decentralized systems may struggle with sub-optimality, as each robot makes decisions based on local information rather than a global view of the system, which can lead to less efficient overall trajectories [5].

2.3 Mixed Integer Non-Linear Programming (MINLP)

Mixed-integer optimization problems appear naturally in many contexts of real-world problems and overcome several new technical challenges that do not appear in the nonmixed counterparts. In addition, MILPs can optimize decisions that take place in complex systems by including integer, binary, and real variables to form logical statements. Mixed-integer linear programs are linear programs composed by linear inequalities (constraints) and linear objective function to optimize with the added restriction that some, but not necessarily all, of the variables must be integer-valued. The term "integer" can be replaced by the term "binary" if the integer decision variables are restricted to take on either 0 or 1 values [2].

Since the nature of the robot's movements and constraints is nonlinear, Mixed Integer Non-Linear Programming (MINLP) is a more widely-used mathematical optimization technique in multi-robot path planning and collision avoidance. MINLP models the robot's movements and constraints, such as obstacle avoidance and task completion, as a set of non-linear equations and inequalities. These constraints often involve both continuous variables (for positions or velocities) and integer variables (for decisions, such as whether a robot should turn or stop) [3].

MINLP can optimize a global objective (such as minimizing travel time or energy consumption) while ensuring that robots do not collide. Its strength lies in the ability to generate globally optimal solutions that take into account the entire system's dynamics. However, the computational complexity of MINLP grows exponentially with the number of robots and constraints, making it more suitable for smaller-scale systems or environments where pre-planning is feasible.

The generical formulation of a MINLP problem can be expressed as [3]:

$$\min_{x} f(x) \quad (1)$$
$$s.t. \begin{cases} c(x) \leq 0 \\ x \in X \\ x_i \in \mathbb{Z}, \forall i \in I \end{cases}$$

Where:

- $f : \mathbb{R}^n \to \mathbb{R}$ and $c : \mathbb{R}^n \to \mathbb{R}^m$ are two continuous functions
- $X \subset \mathbb{R}^n$ is a bounded polyhedrical set
- $I \subseteq \{1, ..., n\}$ is the index set of integer variables

2.4 Reciprocal Velocity Obstacles (RVO)

The Reciprocal Velocity Obstacles (RVO) method is a decentralized approach used for real-time collision avoidance in multi-robot systems [6]. It improves upon the original **Velocity Obstacles (VO)** by considering reciprocal actions between robots. Each robot predicts the future positions of nearby robots and computes a "velocity obstacle", that is a set of velocities that would lead to a collision. By choosing a velocity outside this set, the robot can avoid potential collisions.

Each robot adjusts its velocity based on the predicted movements of others, ensuring that no single robot is solely responsible for avoiding a collision. This approach is efficient and scalable, making it highly applicable to dynamic environments with multiple moving agents. However, RVO can struggle in crowded environments where many robots are operating in close proximity [5]. The mathematical formulation of RVO involves several components:

- Calculation of the velocity obstacle: For each pair of robots i and j with velocities v_i and v_j, the velocity obstacle for agent i relative to agent j can be computed as:

  $$VO_{ij} = \{v_i \in \mathbb{R}^2 \mid d_{ij}(v_i, v_j) < \delta_s\} \tag{2}$$

  Where d_{ij} is the distance between the agents based on their velocity and δ_s is the minimum safety margin.

- Reciprocal velocity adjustment: Each agent i adjusts its velocity by considering the velocity obstacle imposed by other agents:

  $$v_i^* = v_i + \Delta v_i \tag{3}$$

  Where Δv_i is the adjustment required to avoid collision with agent j.

- Combining multiple agents: The overall velocity adjustment can be formulated as an optimization problem, where the goal is to minimize the total disturbance from the desired velocity while ensuring that the adjusted velocities are outside the velocity obstacles:

$$\min \sum_{j \in N} ||v_i^* - v_i||^2 \text{ s.t. } v_i^* \notin VO_{ij} \tag{4}$$

This method allows for real-time computation of safe velocities for multiple robots rimultaneously, making RVO efficient for dynamic environments and suitable for real-time collision avoidance.

2.5 Genetic Algorithms (GA)

Genetic Algorithms (GA) are bio-inspired optimization techniques, modeled after the process of natural selection. In GA, a population of candidate solutions evolves over several generations. Solutions are combined and mutated to produce new offspring, with the best-performing solutions surviving to the next generation. GA can optimize multiple objectives, such as minimizing path length while avoiding collisions [5].

The algorithm begins with a randomly generated population of individuals, each represented by a set of characteristics (called genes) that describe a possible solution. These individuals evolve through an iterative process where the "fittest" individuals are selected based on their ability to solve the problem. Selected individuals undergo genetic operators such as Crossover, Mutation and Elitism to produce new offspring. Over successive generations, the population gradually evolves towards better solutions, as less fit individuals are replaced by more successful ones. The GA terminates when a specified number of generations is reached or when a satisfactory solution is found.

During the evolution process, each individual in the population is evaluated using the objective function, and those with the highest fitness (i.e., those closest to the optimal solution) are more likely to be selected for reproduction, ensuring that over successive generations, the population converges towards better solutions. The following steps are used to obtain fitness [4].

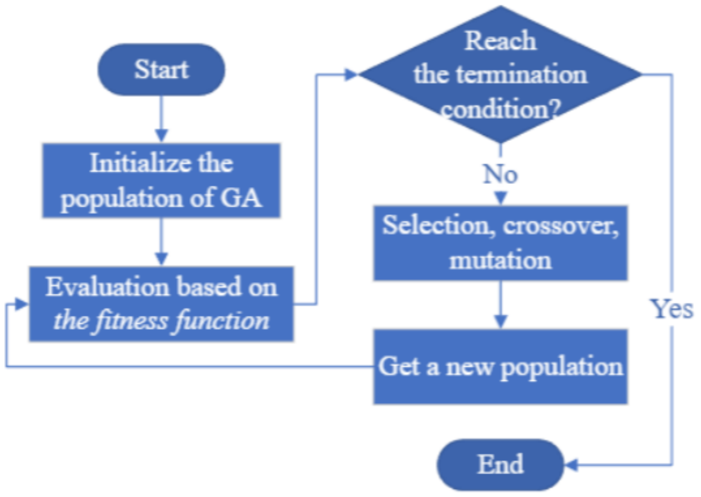

Start
Initialize the population of GA
Evaluation based on the fitness function
Reach the termination condition?
No
Selection, crossover, mutation
Get a new population
Yes
End

Figure 2: Flow-chart of the Genetic Algorithm [5]

1. Consider a population randomly
2. Obtain the fitness of the population
3. Repeat from Step 4 to 6 until convergence
4. Choose parents from the population with a selection criteria based on fitness
5. Generate a new population through crossover, mutation and elitisim
6. Obtain fitness for newly generated populations

Crossover is the process of combining two parent solutions (chromosomes) to produce one or more offspring, with the goal of generating new solutions that inherit characteristics from both parents.
When the Crossover fraction is set to a certain factor C, it means that $C\%$ of the individuals in the population will undergo crossover, while the remaining will either be directly passed on to the next generation, based on the Elitism percentage, or undergo a different genetic operation like mutation.

There exist different types of crossover operators, based on how the new solution is produced starting from the parent solutions. The simplest one is **Uniform Crossover**, that works by treating each gene of the new solution independently and making a random choice as to which parent it should be inherited from. This is implemented by generating a string of l random variables from a uniform distribution over [0,1]. In each position, if the value is below a parameter p (usually 0.5), the gene is inherited from the first parent; otherwise from the second. The second offspring is created using the inverse mapping [7].

Another type of crossover operator is the **Laplace Crossover,** that generates children using either of the formulae 5, chosen at random, where p_1 and p_2 are the parents of c_1 and c_2 and b is a random number generated from a Laplace distribution [7] [8] [9].

$$\begin{aligned} c_1 &= p_1 + b \cdot |p_1 - p_2| \\ c_2 &= p_2 + b \cdot |p_1 - p_2| \end{aligned} \tag{5}$$

Figure 3 shows an example with $b = 1/2$, for which the two potential offsprings are identical.

Mutation introduces random alterations to the genes of individuals in the population, thereby increasing the diversity of the population and preventing premature convergence to a local optimum.
Like Uniform Crossover, Uniform Mutation generates one new solution from one single parent, by altering its genes with a probability defined by the Mutation Rate. If the Mutation Rate is R, the genes of the parent solution have an $R\%$ probability of being mutated to a new value, randomly selected within the bounds of the solution with a uniform distribution [7]. Mutation helps the algorithm explore new areas of the search space by introducing small, random changes to offspring generated during crossover.

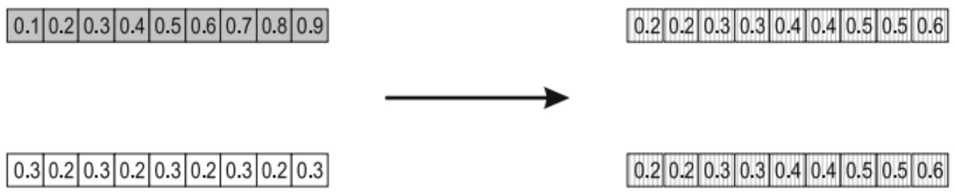


Figure 3: Laplace crossover with $b = 1/2$. From [7], figure 4.8.

2.6 Summary of approaches

MINLP is a mathematical optimization framework adept at handling problems characterized by both discrete and continuous variables, as well as nonlinear relationships. This versatility makes it particularly suitable for complex systems where decision variables are interdependent and constraints are nonlinear. However, the computational intensity of MINLP can be a significant drawback, especially for large-scale problems. GAs are effective in exploring vast and complex search spaces, making them suitable for problems with numerous variables and potential solutions. Their flexibility allows adaptation to various optimization problems without requiring gradient information. However, their performance is sensitive to parameter settings, necessitating careful tuning to achieve optimal results.

The RVO approach is a real-time collision avoidance method used in multi-agent navigation. It enables each agent to independently navigate while accounting for the reactive behaviors of other agents, ensuring safe movements. RVO is particularly effective in dynamic environments where multiple agents operate in close proximity. However, its application is primarily focused on collision avoidance rather than optimization of broader logistical operations.

In this study, the problem will be formulated as a MINLP optimization problems and solved using GA. The choice of MINLP relies in its ability to model the complexities of logistics problems like cooperative transportation operations, which involve both continuous and integer variables as well as nonlinear constraints, enabling a detailed and accurate description of the dynamics in transportation operations.

To solve these MINLP problems, GA is employed due to its suitability for tackling complex and multi-dimensional optimization challenges without relying on gradient information. Unlike gradient-based methods, GA uses a population-based search strategy to iteratively evolve and improve solutions over generations [8]. This approach is particularly advantageous in scenarios where exact gradients are challenging to compute. Furthermore, GA's flexibility in managing multiple objectives makes it an excellent choice for optimizing ground transport logistics, where the goals include minimizing travel time and energy consumption.

While real-time methods, such as RVO, are valuable for online collision avoidance in dynamic settings, they do not offer the same level of optimality for pre-planned, complex operations as an offline optimization approach does. Offline optimization allows to analyze and optimize transporter paths in advance, ensuring that the planned operations are as efficient as possible before implementation [10]. This is especially advantageous in a controlled industrial setting where transport operations can be planned ahead of time.

3 Introduction

This chapter introduces the proposed approach for solving the multi-robot collision avoidance and optimization problem, starting with key definitions and concepts required to frame the problem, including the environment in which the robots operate, the nature and the constraints of their trajectories. The chapter will then present the problem statement, describing the main challenges that need to be overcome, and set the stage for the optimization methods that will be employed in the subsequent chapters.

3.1 Definitions

A system of n robots that move in a two-dimensional environment is considered. One path for each of the n robots is produced through the CDT-RRT* [11], discussed in section 4.1, that is already guaranteed to be collision-free with respect to the static obstacles. The single path is defined as a set of points connecting an initial point $[x_i, y_i]$ to a final point $[x_f, y_f]$.

Each robot is associated with a trajectory $p_i(t_k) = \big[x_i(t_k), y_i(t_k)\big]$ sampled at each time-step t_k with $k = 1, ..., N_i$, where N is the number of samples. The trajectory, produced by the trajectory planning algorithm described in section 4.2, is travelled by the robot with a velocity $v_i(t_k)$ and an acceleration $a_i(t_k)$, subject to the following constraints:

$$\begin{aligned} |v_i(t_k)| &\leq v_{max} \\ |a_i(t_k)| &\leq a_{max} \end{aligned} \tag{6}$$

Each robot completes their whole trajectory in a time T_i. Besides, the robots are supposed to all have the same mass m and the same diameter.

A collision is detected between robot i and j at time-step t_k if the distance between the two robots at time-step t_k is below the safety-margin δ_s:

$$||\, p_i(t_k) - p_j(t_k) \,|| \leq \delta_s \tag{7}$$

$$\begin{aligned} k \in \{1, ..., \min(N_i, N_j)\} \\ i, j \in \{1, ..., n\}, i \neq j \end{aligned}$$

3.2 Problem statement

The goal is to develop a trajectory optimization algorithm that adjusts the velocities of the robots to prevent collisions, without altering the original paths. The system must consider the following factors:

1. The algorithm must ensure that the robots respect the safety margin during their movement by modifying their velocities based on their kinematic constraints.

2. The optimization must balance multiple objectives, including minimizing total travel time, minimizing energy consumption, and ensuring a safety margin between the robots during motion. The optimization should handle these conflicting objectives while maintaining feasible solutions.

To achieve these objectives, this work will begin by analyzing a Brute-force approach and a Prioritized planning approach to identify the best combination of velocity profiles.

Following that, the focus will be moved on the usage of Genetic Algorithms to solve MINLP problems, aimed at optimizing the velocity profiles of each robot offline. The initial step will involve solving a single-objective problem aimed only at minimizing the total task completion time. Subsequently, the analysis will be extended to a multi-objective problem that also considers the total energy consumption during task execution.

4 Single-robot trajectory generation

This chapter focuses on the generation of single-robot trajectories, by discussing first path generation and then moving on to trajectory planning, where the analysis moves from path-finding to how the robot moves along the path over time. Then, different interpolation techniques will be introduced that allow for smooth velocity profiles. This chapter forms the basis for how each robot individually navigates its environment, before considering multi-robot interaction.

4.1 Path generation

A path is defined as a route that passes through a series of predefined waypoints, starting from an initial point and leading to a final destination. These via-points are essentially a set of coordinates that lie within the boundaries of the map representing the environment in which the robot operates. The role of these points is to guide the robot along its trajectory, ensuring that it navigates through the specified area.

To compute this route, a path planning algorithm is used. This type of algorithm generates a path from the initial to the final point that the robot is expected to follow, taking into account the spatial configuration of the environment, possible obstacles, and sometimes additional constraints. A crucial aspect of path planning is that the algorithm focuses only on the geometry of the route. However, the path itself does not include temporal information, such as the exact time at which the robot should cross each via-point. The timing or velocity profile along the path is typically handled separately by trajectory planning algorithms, which define how fast the robot should move at each point in time. The distinction between path planning and trajectory planning is important, as it separates the geometry of the route from the temporal constraints of the robot's movement.

The path planning algorithm employed in this work is the **CDT-RRT***, a variation of the Rapidly-exploring Random Tree algorithm, which is enhanced by **Constrained Delaunay Triangulation (CDT)**. This technique leverages the mathematical properties of Delaunay Triangulation to divide the environment's map into a mesh of triangles. The key principle behind Delaunay Triangulation is that, given a set of points, it constructs triangles by connecting points with edges in such a way that no point lies inside the circumcircle of any triangle. This method ensures that the triangulation is optimal in terms of

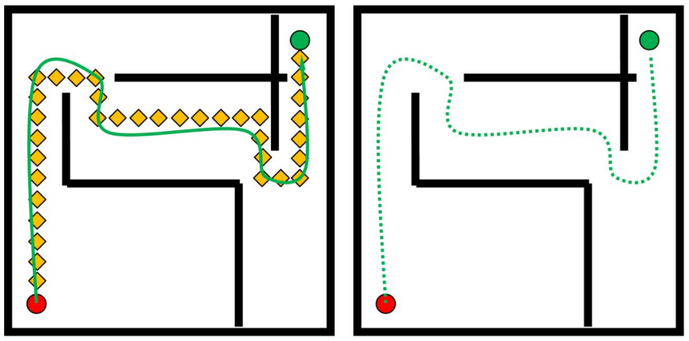


Figure 4: A generic path and its interpolated trajectory.

minimizing the maximum angle of the triangles, which provides a more stable and efficient structure for pathfinding.

By covering the entire map with a network of triangles, the CDT serves as the foundation for the CDT-RRT* algorithm, which is responsible for generating paths between a start point and a goal point. The CDT-RRT* algorithm integrates the random sampling-based exploration of the RRT* method with the Delaunay Triangulation, offering a robust way to explore the environment while ensuring optimal path-finding.
The result is a path planning algorithm that generates a feasible path and also optimizes the path in terms of metrics like travel distance. By taking advantage of the triangulation network, the algorithm efficiently explores the environment, adapting to various shapes and constraints within the map, and identifying an optimal route that connects the initial and goal points while navigating through obstacles.

The map of the environment is composed of a set of lines, each identified by a pair of points with continuous coordinates. These lines define the boundaries and structure of the environment. When these lines form a closed shape, they represent an obstacle. Obstacles may vary in size and shape, depending on the specific environment in which the robot operates, and these closed sets of lines essentially delineate regions that the robot cannot traverse.

The position of the robot within this environment is referred to as the robot's configuration. This configuration defines how the robot is situated relative to the environment and its obstacles. The set of all possible configurations that the robot can assume forms what is known as the configuration space

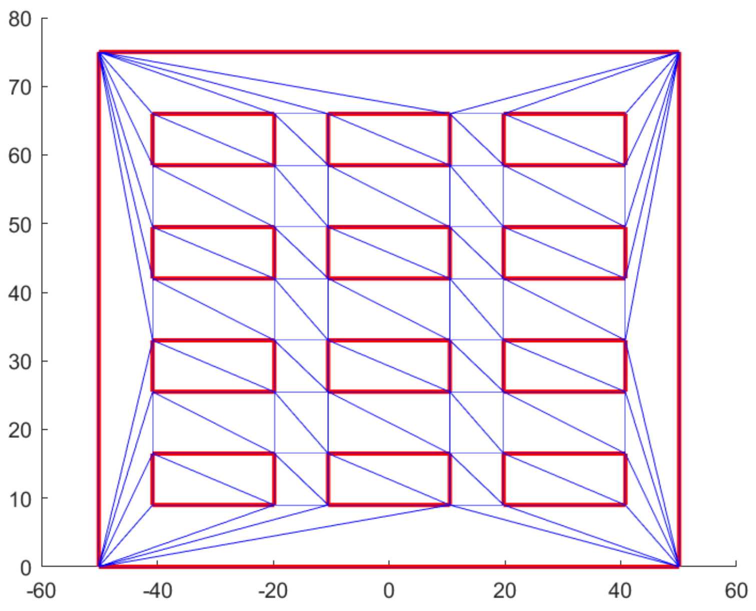


Figure 5: Representation of the map of the environment with Delaunay Triangulation applied.

(or C-space). In the context of path planning, the configuration space is a crucial concept because it allows the robot's movement to be mapped in a more abstract manner, taking into account both its physical position and the constraints imposed by obstacles in the environment.

Within the configuration space, two distinct regions exist: the obstacle configuration space and the obstacle-free configuration space. The obstacle configuration space represents the set of all configurations in which the robot's position causes it to intersect or collide with any obstacle. This space is essentially a projection of the obstacles into the robot's possible configurations, creating regions in the configuration space that are blocked or inaccessible. In contrast, the obstacle-free configuration space is the set of all configurations where the robot can move freely without intersecting any obstacle. This space is crucial for path planning, as it defines where the robot is able to safely navigate without risk of collision.

A map of the environment can be imported into MATLAB by starting with a CAD file that contains the geometric representation of the environment. Once the CAD file is imported, it can be converted into a format that MATLAB can process, as a set of lines and points that fully describe the map. On this map, CDT can be applied as shown in figure 6.

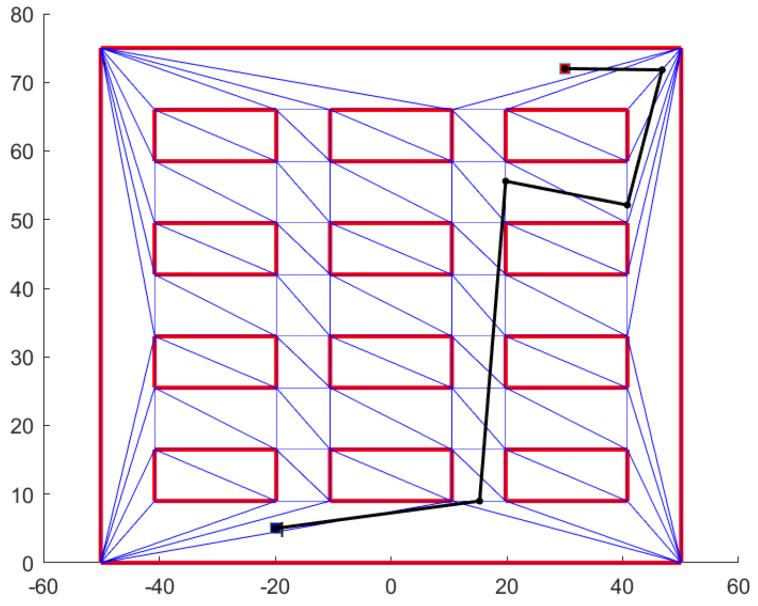


Figure 6: An example of a path generated through the CDT-RRT* algorithm.

4.2 Trajectory planning

In path planning algorithms, the focus is primarily on determining a feasible geometric route through the environment, and as a result, the time is not considered. These algorithms concentrate on finding a collision-free path from a starting point to a goal point, ensuring that the robot avoids obstacles along the way. However, while this geometric path is important, it does not account for the temporal aspects of the robot's movement. For a robot to navigate effectively in a real-world setting, it is not enough to simply find a valid route, it is equally critical to consider how the robot moves along this path over time.

The role of trajectory planning is to convert the geometric path into a time-parameterized sequence of positions, velocities, and accelerations [12]. The goal is to generate the reference inputs that the robot's control system will use to ensure that it follows the desired path in a feasible manner. This involves generating a time sequence of the robot's states, such as positions, velocities, and accelerations along the path.

An interpolating function is employed to define the desired trajectory in a smooth and continuous way. These functions are often polynomials, as they allow for the smooth interpolation of position data, and their derivatives can be used to compute velocities and accelerations. Polynomial interpolation provides flexibility in shaping the trajectory while ensuring that velocity and acceleration profiles are consistent with the robot's limits.

4.3 Interpolation

Interpolation is a method of estimating values between known data points. It can be used to generate additional points between the points generated by the CDT-RRT* so that, in a second step, these points can be associated with a time sequence which then generates a trajectory.
There are several types of interpolation that can be used for this purpose [13]:

- Piecewise linear interpolation
- Cubic spline interpolation
- Shape-preserving Piecewise cubic Hermite interpolation

Piecewise linear interpolation generates a continuous function that interpolates the set of data points, called breakpoints, of the path. The first derivative of

this interpolating function is not continuous and so it gives a velocity law that is not continuous.
To make both the first and second derivatives of the interpolating function continuous, the cubic spline interpolation type can be used. But again, the problem with this method is that it does not preserve the shape of the given path, because in order to keep the second derivative also continuous, some overshoots are generated with respect to the shape of the path.

A compromise between the first and second types of interpolation is the Piecewise cubic Hermite interpolation, because it returns an interpolation function whose second derivative is discontinuous, but the shape of the path is preserved without any overshoot. Figure 7 shows the differences between the different types of interpolation [13], while figure 8 shows the two types of cubic inerpolation applied to the path.

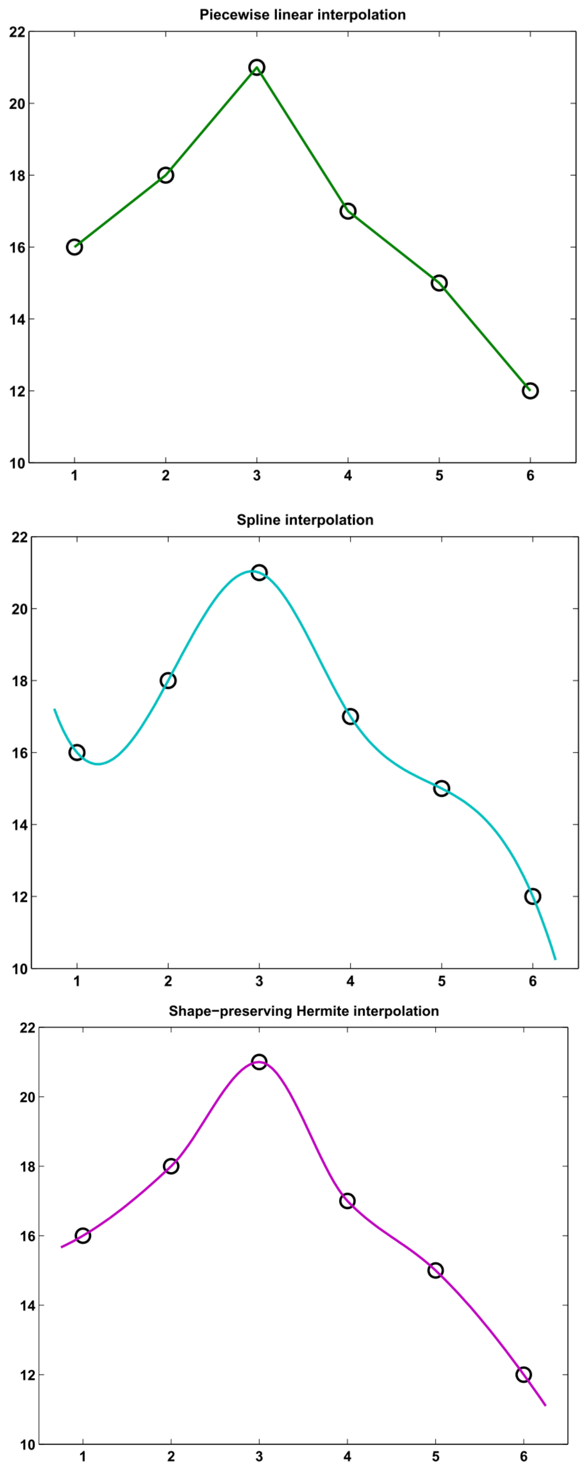


Figure 7: Difference between the different types of interpolations [13].

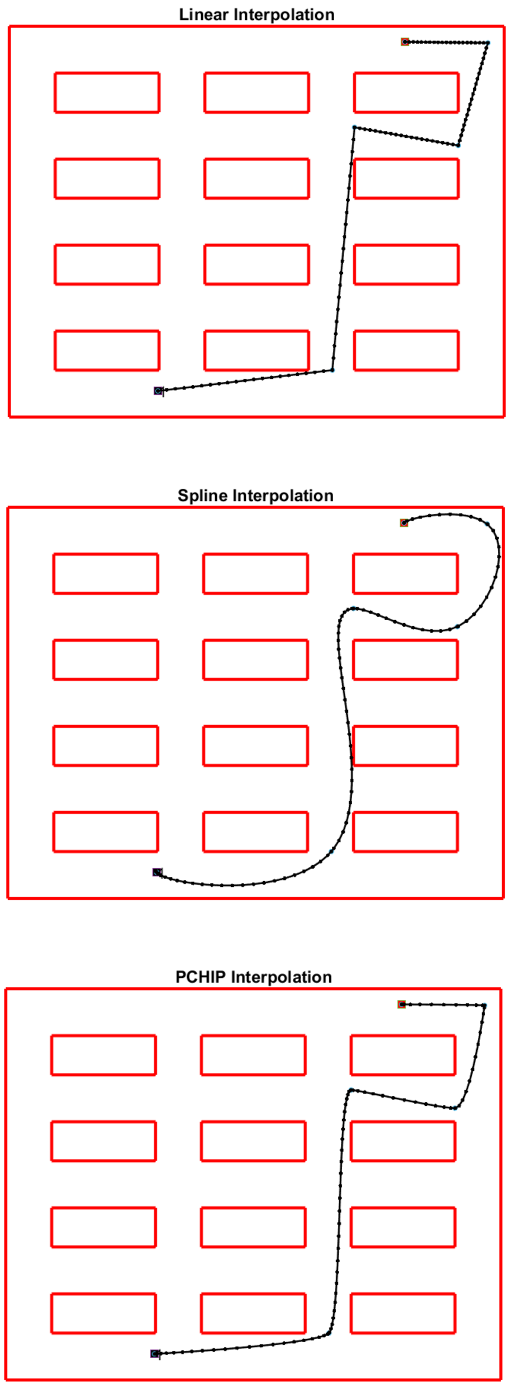


Figure 8: Different types of interpolation applied to the path generated by the CDT-RRT* algorithm in figure 6, with 100 samples.

The result of the PCHIP interpolation are the velocity and acceleration profiles shown in figures 13 to 16. This profile exhibits a smooth velocity transition, where the robot starts with zero velocity and increases it, until reaching the maximum velocity allowed by the system's kinematic constraints, around the midpoint of the segment.

Once the robot has achieved this maximum velocity, it starts to decelerate as it approaches the end of the segment. The deceleration ensures that the robot reduces its speed appropriately before entering the next segment, allowing for better control and stability during transitions.

The final velocity at the end of each segment is influenced by the angle between the current segment and the next. If the angle is sharp, the robot's final velocity before entering the next segment is lower. This is to ensure that the robot can navigate the turn safely without overshooting or destabilizing, respecting its dynamic constraints. On the other hand, for larger, less abrupt angles, the robot can maintain a higher speed at the end of the segment, leading to a more efficient trajectory. This approach ensures both smooth navigation and the compliance to the robot's dynamic limitations.

5 Multi-robot trajectory optimization

This chapter introduces the core aspect of this work: multi-robot trajectory optimization. It builds on the single-robot trajectory generation discussed in the previous chapter by coordinating multiple robots in shared environments.

The chapter begins with a discussion on finish time optimization, outlining a brute-force approach for determining velocity combinations that prevent collisions. It then explores more efficient strategies, such as prioritized planning, where robots are assigned priorities to sequentially avoid collisions. The focus then shifts to more advanced techniques like maximum velocity optimization, where robots' velocities are optimized simultaneously, ensuring collision avoidance without unnecessary slowdowns. The chapter also introduces a slow-down segment optimization strategy that focuses on decelerating robots only at the beginning of their trajectories to avoid collisions, minimizing disruption to the overall travel time. Finally, this chapter concludes by addressing multi-objective optimization, which seeks to balance minimizing both finish time and energy consumption, with a detailed exploration of the Pareto front and how the optimal solution can be selected.

5.1 Finish time optimization

5.1.1 Brute-force approach

The first implemented approach involves a systematic exploration of potential velocity combinations for the n robots. Initially, the algorithm computes all possible combinations of maximum velocities for each robot. It then evaluates these combinations to identify those that prevent collisions by ensuring that no two robots occupy the same space at the same time.

Once the collision-free velocity combinations are determined, the algorithm selects the best combination that minimizes the variance between the actual velocities and a specified target maximum velocity. This approach balances the need for collision-free trajectories with the goal of maintaining velocities as close as possible to the desired target maximum velocity, hence optimizing the arrival time of each robot.

A collision-free combination of maximum velocities is identified by a vector $\bar{v}_i = [v_{i_1}, v_{i_2}, ..., v_{i_n}]$, with $i \in \{1, ..., c\}$ and c the number of combinations.

Then, the following criterion is defined:

$$f(\bar{v}_i) = \alpha \sum_{j=1}^{n} |v_{max} - \bar{v}_{i_j}| + \beta\, \sigma(\bar{v}_i) \tag{8}$$

That has to be minimized to achieve the above task and find the best collision-free velocity combination v^*:

$$v^* = \min_{\bar{v}_i} f(\bar{v}_i) \tag{9}$$

The possible combinations of maximum velocities v_i can be computed from a range between a set minimum velocity v_{min} and the maximum velocity v_{max}. Since the set of possible velocities must be discrete, a velocity step d must be chosen. The number of possible combinations is given by:

$$c = (\frac{v_{max} - v_{min}}{d} + 1)^n \tag{10}$$

The main issue with this approach is that the number of combinations to check increases exponentially with the number of robots, and so does the execution time of the algorithm.

Figure 9 shows the flow-chart of this algorithm, where v_i are all the possible combinations of maximum velocities for the n robots, given v_{max}, v_{min} and d, while $\bar{V}$ is the set of collision-free combinations.

To significantly reduce the execution time of the algorithm, different levels of velocity combinations are considered. The algorithm starts from the highest possible value of v_{min}, and then reduces its value until collision-free velocity combinations are found. A practical example is described in section 6.3.

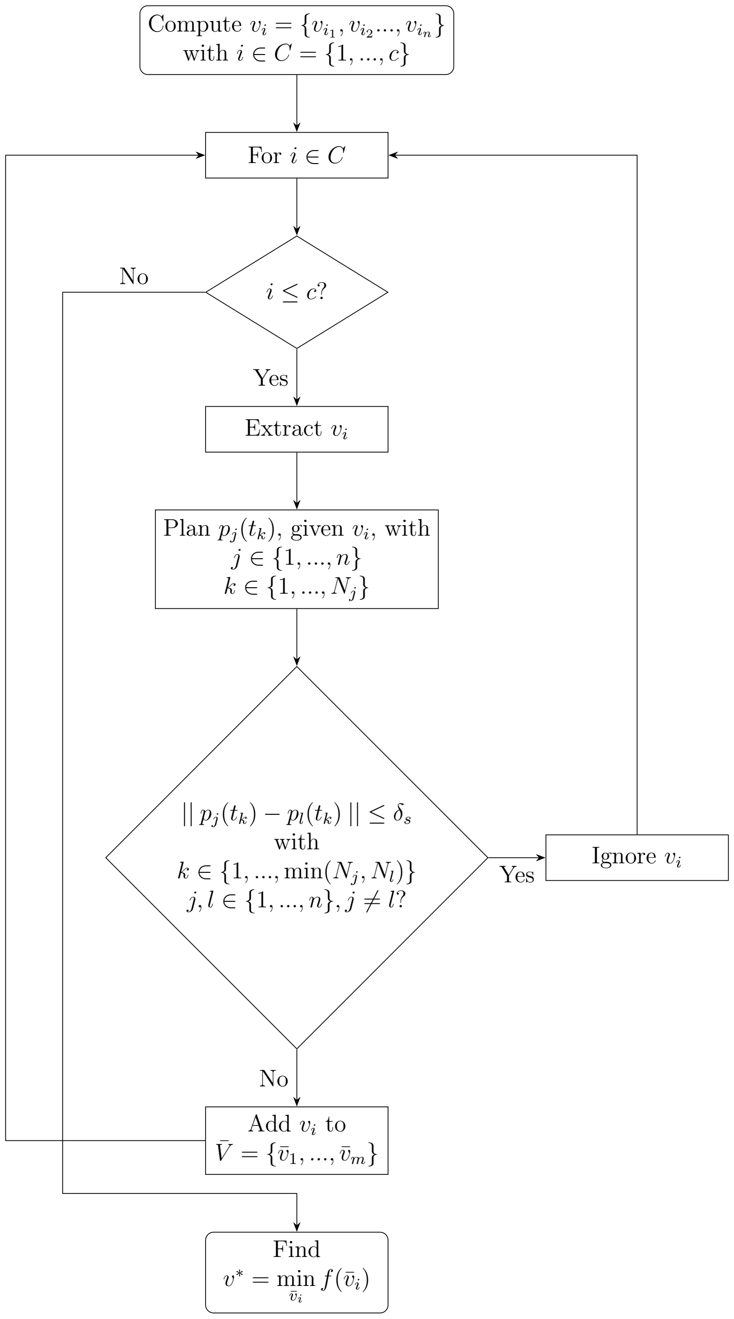

Compute $v_i = \{v_{i_1}, v_{i_2}..., v_{i_n}\}$ with $i \in C = \{1, ..., c\}$
For $i \in C$
$i \leq c$?
No
Yes
Extract v_i
Plan $p_j(t_k)$, given v_i, with $j \in \{1, ..., n\}$ $k \in \{1, ..., N_j\}$
$||\, p_j(t_k) - p_l(t_k)\, || \leq \delta_s$ with $k \in \{1, ..., \min(N_j, N_l)\}$ $j, l \in \{1, ..., n\}, j \neq l$?
Yes
Ignore v_i
No
Add v_i to $\bar{V} = \{\bar{v}_1, ..., \bar{v}_m\}$
Find $v^* = \min_{\bar{v}_i} f(\bar{v}_i)$

Figure 9: Algorithm for the Brute-force approach

5.1.2 Prioritized planning

The idea of checking all possible combinations can make the execution slow, and so considerations for more efficient methods have been performed.

An approach to motion planning for multiple robots is prioritized planning. In a prioritized approach each of the robots is assigned a priority. Then in order of decreasing priority, the robots are picked. For each picked robot a trajectory is planned, avoiding collisions with the previously picked robots, which are considered as dynamic obstacles. This reduces the multi-robot motion planning problem to the problem of motion planning for a single robot in a known dynamic environment, which is a difficult problem in itself [14].

An important question is how to assign the different priorities to the robots. There can of course be natural reasons to assign different priorities, for instance based on the importance of the tasks or on different starting times of the robots, but we do not assume that in this work because all the robots collaborate to achieve the same task.

If we aim to minimize the maximum of the arrival times, robots that have to traverse long distances (and hence need much time) should be able to do this relatively unhindered, while robots that have to traverse short distances can afford to spend time on avoiding robots with higher priority. For this reason, the chosen heuristic is the length of the path of the robot.

Once priorities are assigned, a trajectory is planned for the highest priority robot (the one with the longest path). Then, the trajectory for the second robot is planned, avoiding collision with the first robot. To do so, its maximum velocity is decreased by a decreasing step d, until there is no collision.
Then, the trajectory is planned for the third robot, avoiding collision with the first two, and so on. It is important to notice that also in this case the decrement of maximum velocities is sampled and not continuous, in order to reduce the execution times.

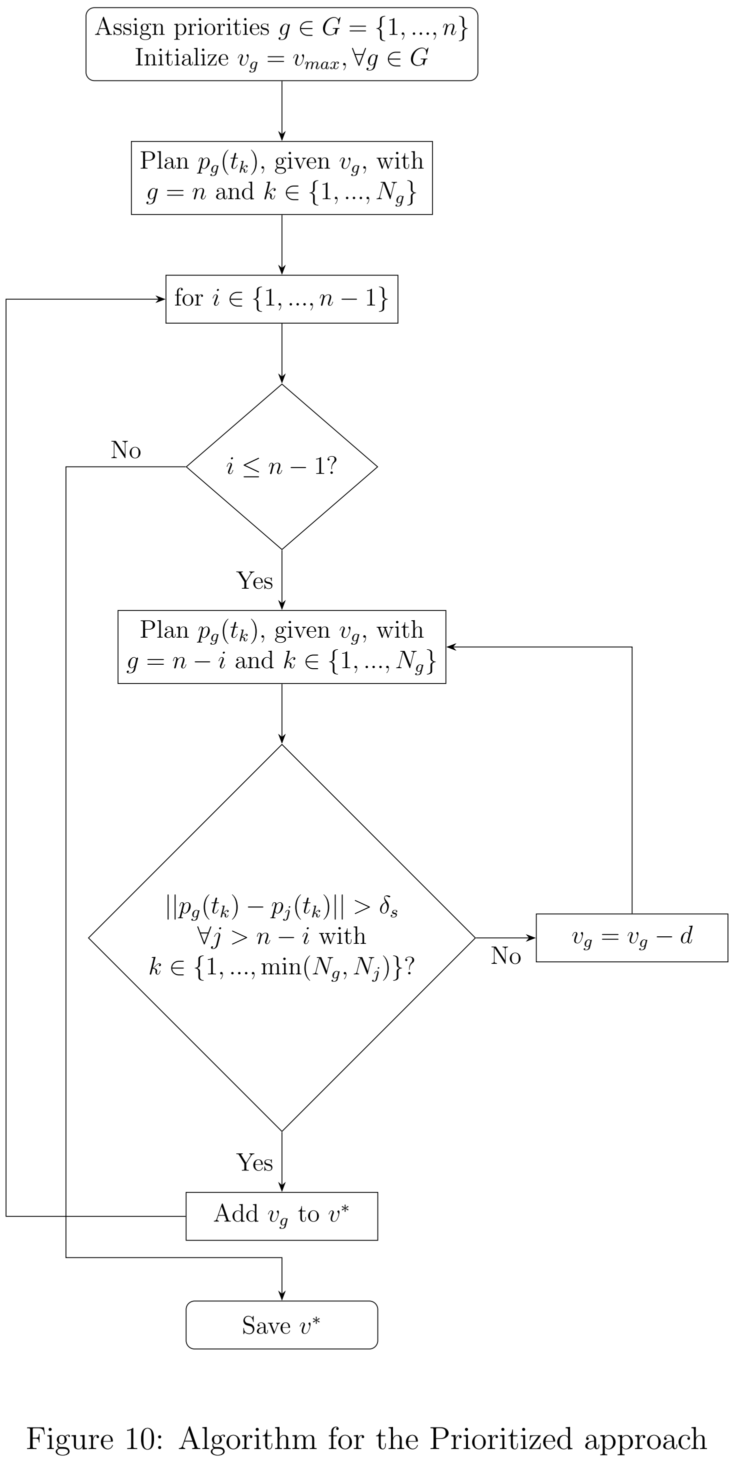

Assign priorities $g \in G = \{1, ..., n\}$
Initialize $v_g = v_{max}, \forall g \in G$
Plan $p_g(t_k)$, given v_g, with $g = n$ and $k \in \{1, ..., N_g\}$
for $i \in \{1, ..., n-1\}$
$i \leq n-1$?
No
Yes
Plan $p_g(t_k)$, given v_g, with $g = n - i$ and $k \in \{1, ..., N_g\}$
$||p_g(t_k) - p_j(t_k)|| > \delta_s$ $\forall j > n - i$ with $k \in \{1, ..., \min(N_g, N_j)\}$?
No
$v_g = v_g - d$
Yes
Add v_g to v^*
Save v^*

Figure 10: Algorithm for the Prioritized approach

5.2 Maximum velocities optimization

To overcome the limitations of the Brute-force approach and Prioritize planning, an approach that directly optimizes the maximum velocities of each robot's trajectory was implemented. In this method, the primary goal is to minimize the total travel time for all robots, while ensuring collision avoidance. This is achieved by setting the collision-avoidance conditions as constraints in the optimization problem.

In this approach, instead of iterating through all possible combinations of velocities and checking for collisions, the optimization problem is formulated with the robots' maximum velocities as decision variables. The optimization objective is to minimize the total travel time, which is defined as the time taken for the last robot to reach the goal. At the same time, collision avoidance is guaranteed by setting constraints that prevent robots from occupying the same space at the same time.

These collision constraints take into account the trajectories of all robots and ensure that the solution respects the physical limitations of each robot. Specifically, the optimization problem ensures that no two robots are at the same position within a critical distance during their movement. If a potential collision is detected during optimization, the velocity of the robot involved is adjusted to ensure that it remains at a safe distance from others.

Additionally, the optimization approach incorporates the kinematic constraints of each robot, ensuring that the chosen velocities stay within their physical capabilities. By solving this optimization problem, the algorithm produces the optimal combination of maximum velocities for all robots that achieves both collision avoidance and minimum task completion time.

The optimization problem is defined as follows:

$$\min_{v_i} \max T_i(v_i) \tag{11}$$

$$\text{s.t.} \quad \begin{cases} ||p_i(t_k) - p_j(t_k)|| \geq \delta_s, \forall i \neq j, \forall t_k \\ v_{min} \leq v_i \leq v_{max} \end{cases}$$

$$\text{with: } \begin{aligned} & i, j \in \{1, ..., n\} \\ & k \in \{1, ..., N_i\} \end{aligned}$$

In contrast to prioritized planning, which assigns priorities to robots based on path length and sequentially adjusts velocities to avoid collisions, this method allows for simultaneous optimization of all robots' velocities. This ensures a more balanced solution, where no robot is unnecessarily slowed down, leading to improved efficiency in terms of total travel time.

5.3 Slow-down segment optimization

An approach that introduces a localized slow-down segment at the beginning of each robot's trajectory has been implemented. This method focuses on strategically slowing down robots only during the initial part of their paths to prevent collisions, while allowing them to operate at their maximum allowable velocities for the rest of their trajectories. The idea is to minimize overall travel time by solving potential collisions early in the movement, avoiding the inefficiency of reducing velocities throughout the entire path.

In this approach, the robots' trajectories are divided into two distinct phases:

1. In a first phase, robots are decelerated at the start of their movement to ensure safe navigation around potential collision points. This initial slowdown allowes to avoid to modify velocities throughout the entire trajectory.
2. After passing through the slow-down segment, robots return to their maximum velocities, continuing along the remaining portion of their path without further adjustments.

By decelerating robots in the early part of their journey, the potential for collisions is reduced, and robots can proceed at full speed thereafter, minimizing disruptions to their overall efficiency.

To implement this method, the problem was modeled as a MINLP problem (section 2.3). The objective is to find the optimal length and velocity scaling factor for the slow-down segment of each robot's trajectory, ensuring collision avoidance while minimizing the total task completion time.
The optimization problem is defined as follows:

- **Objective Function**: The goal to minimize the total travel time for all robots is achieved through the following objective function:

$$f(L_{s_i}, \alpha_i, d_i) = \max T_i(L_{s_i}, \alpha_i, d_i) \tag{12}$$

- **Decision Variables**: The primary decision variables are:
  - The length of the slow-down segment for each robot L_{s_i}.
  - The velocity scaling factor applied during the slow-down segment α_i.
  - The binary selection variable d_i, where $d_i = 1$ if the slow-down segment is applied to the trajectory of robot i, and $d_i = 0$ it it is not applied.
- **Constraints**: To ensure the feasibility of the solution, the optimization is subject to:
  1. **Collision avoidance constraints**: These constraints ensure that robots maintain a safe distance from each other during the slow-down segment, preventing collisions at potential intersections. This is achieved by modeling the robot positions and velocities over time and enforcing a minimum distance constraint
  2. **Kinematic constraints**: These constraints ensure that the deceleration applied in the slow-down segment adheres to the robots' kinematic limits, such as maximum and minimum velocities and acceleration rates.

The overall optimization problem is modeled as follows:

$$\min_{L_{s_i},\alpha_i,d_i} f(L_{s_i},\alpha_i,d_i) \tag{13}$$

$$\text{s.t.} \quad \begin{cases} ||p_i(t_k) - p_j(t_k)|| \geq \delta_s, \forall i \neq j, \forall t_k \\ 0 \leq L_{s_i} \leq L_{i,max} \\ \alpha_{min} \leq \alpha \leq 1 \\ d_i \in \{0,1\} \end{cases}$$

$$\text{with: } i,j \in \{1,...,n\}$$
$$k \in \{1,...,N_i\}$$

The MINLP formulation is essential because the problem involves both continuous and discrete variables, making it highly complex. The continuous variables include the velocities and segment lengths, while the discrete variables represent binary decisions, such as whether a robot should decelerate and for how long. Additionally, the problem is nonlinear due to the dependence of travel time and collision risks on the velocities and trajectories, which introduce nonlinear constraints.

This approach offers some key advantages:

1. By focusing the deceleration on a small part of the trajectory, robots can maximize their velocity for the majority of their path, improving overall task efficiency.
2. The slow-down segment allows for strategic velocity adjustments that prevent collisions without requiring extensive modification of the robots' entire trajectories, that could lead to extreme delays.

By integrating this slow-down segment approach into the broader optimization framework, the robots can collaborate efficiently while avoiding collisions, resulting in a solution that is both time-optimal and collision-free.

5.4 Finish time and energy consumption optimization

Beyond simply minimizing the total completion time, energy efficiency is another critical factor, especially in environments where robots must operate over extended periods or when power consumption is a limiting resource. Therefore, a multi-objective optimization approach that simultaneously considers both the minimization of energy consumption and the minimization of the total task completion time was implemented.

Reducing energy usage extends operational time, reduces wear on components, and lowers operational costs, making it a key performance indicator in multi-robot planning. However, minimizing energy consumption often involves lowering speeds, which directly conflicts with the objective of minimizing task completion time. The goal of minimizing the total task completion time typically leads to higher velocities, which results in faster robot movements. However, increasing velocities often requires more energy, especially when accelerating or moving against resistive forces. Conversely, minimizing energy consumption generally requires the robots to move at lower speeds, which extends their travel times. This creates a natural trade-off between the two objectives: optimizing for speed tends to increase energy consumption, while optimizing for energy efficiency tends to lengthen the completion time [15].

To address this trade-off, a multi-objective optimization framework is necessary, where the algorithm aims at finding a balance between the two goals. The challenge is in finding solutions that find the best compromise between minimizing both objectives while respecting the collision-avoidance and kinematic constraints [16].

5.4.1 Pareto front and solution selection

In multi-objective optimization, there is rarely a single best solution that satisfies all objectives simultaneously. Instead, the algorithm generates a set of Pareto-optimal solutions, which form what is known as the Pareto front. A solution is Pareto-optimal if no other solution can improve one objective without worsening another. In other words, each solution on the Pareto front represents a trade-off: improving the completion time would increase energy consumption and vice versa.

The Pareto front demonstrates the trade-offs between the two objectives. Each

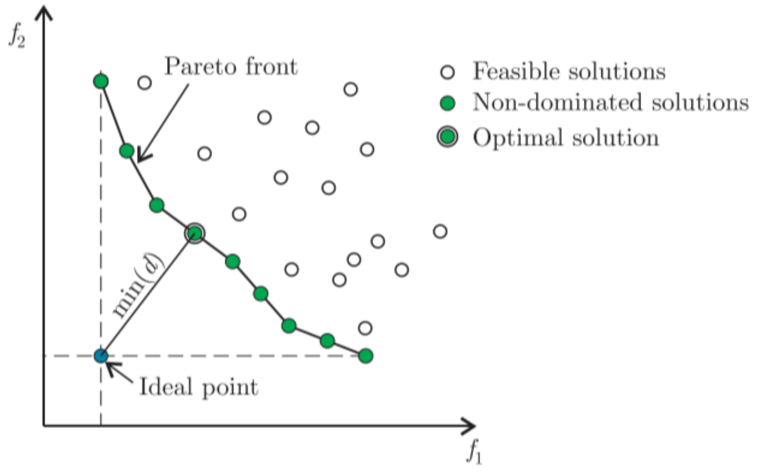


Figure 11: Schema of the Pareto front, the ideal point, and the final optimal solution for the minimization of two contradictory objectives f_1 and f_2 [16].

point on the front represents a potential solution, with different levels of compromise between energy efficiency and completion time. This allows decision-makers to choose a solution based on which objective they prioritize. For example:

- If completion time is the primary objective, a point on the Pareto front with minimal task duration but higher energy consumption would be selected.
- On the other hand, if energy efficiency is more critical, a solution with lower energy consumption but a longer task duration would be preferred.

The multi-objective optimization problem is modeled as:

$$\min_{L_{s_i},\alpha_i,d_i} [\max T_i(L_{s_i},\alpha_i,d_i), \sum_{i=1}^{n}\sum_{k=1}^{N_i} m_i a_i(t_k) v_i(t_k)\Delta t_k] \tag{14}$$

$$\text{s.t.} \begin{cases} ||p_i(t_k) - p_j(t_k)|| \geq \delta_s, \forall i \neq j, \forall t_k \\ 0 \leq L_{s_i} \leq L_{i,max} \\ \alpha_{min} \leq \alpha \leq 1 \\ d_i \in \{0, 1\} \end{cases}$$

$$\text{with: } \begin{aligned} & i, j \in \{1, ..., n\} \\ & k \in \{1, ..., N_i\} \end{aligned}$$

Where $a_i(t_k)$ and $v_i(t_k)$ are the acceleration and the velocity of the robot i at the time-step k, $\Delta t_k = t_k - t_{k-1}$ and m_i is the mass of the robot i.

6 Simulations results

6.1 Setup

The simulations are carried out in a Matlab-based environment, defined on a map as introduced in Section 4. The setup consists of seven robots, each starting from a unique initial position but converging towards the same goal point. The trajectories of the robots are illustrated using solid lines, each in a distinct color to represent a specific robot. Trajectory generation is performed using the PCHIP algorithm, detailed in Section 4.3, according to uniform kinematic constraints for all robots, as outlined in Table 1. The initial configuration of the scenario is depicted in Figure 12, where collision points are marked with empty circles at their respective coordinates.

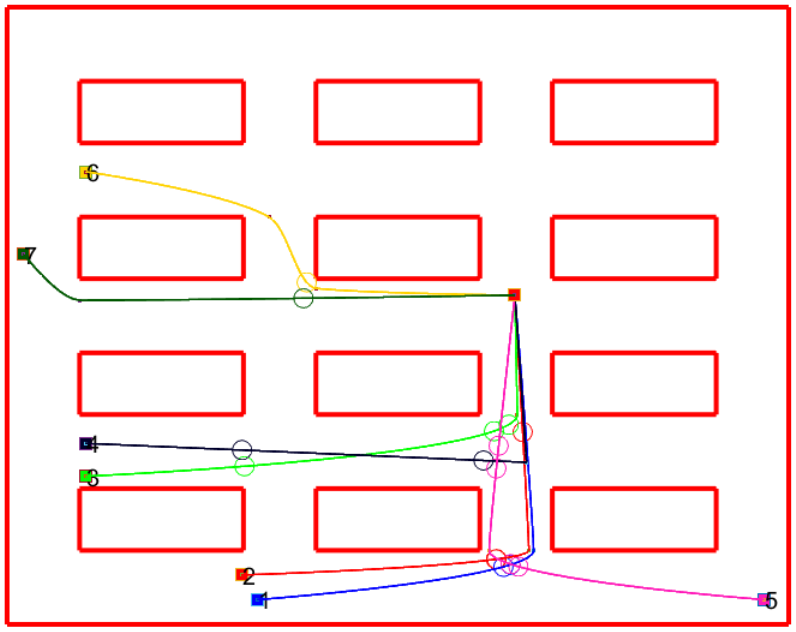


Figure 12: Starting map and scenario for the simulations, including the generated trajectories of seven robots, colliding with other robots at the coordinates marked by the empty circles.

Parameter	**Value**
Map size	$100 \times 75\,\mathrm{m}$
Number of robots	7
Starting points	$\begin{bmatrix} -18 & 3 \\ -20 & 6 \\ -40 & 18 \\ -40 & 22 \\ 47 & 3 \\ -40 & 55 \\ -48 & 45 \end{bmatrix}$
Goal point	$\begin{bmatrix} 15 & 40 \end{bmatrix}$
Robots radius	$1\,\mathrm{m}$
Maximum velocity	$1.5\,\mathrm{m/s}$
Maximum acceleration	$1\,\mathrm{m/s}^2$
Safety margin	$2\,\mathrm{m}$

Table 1: Outline of the parameters used for simulations.

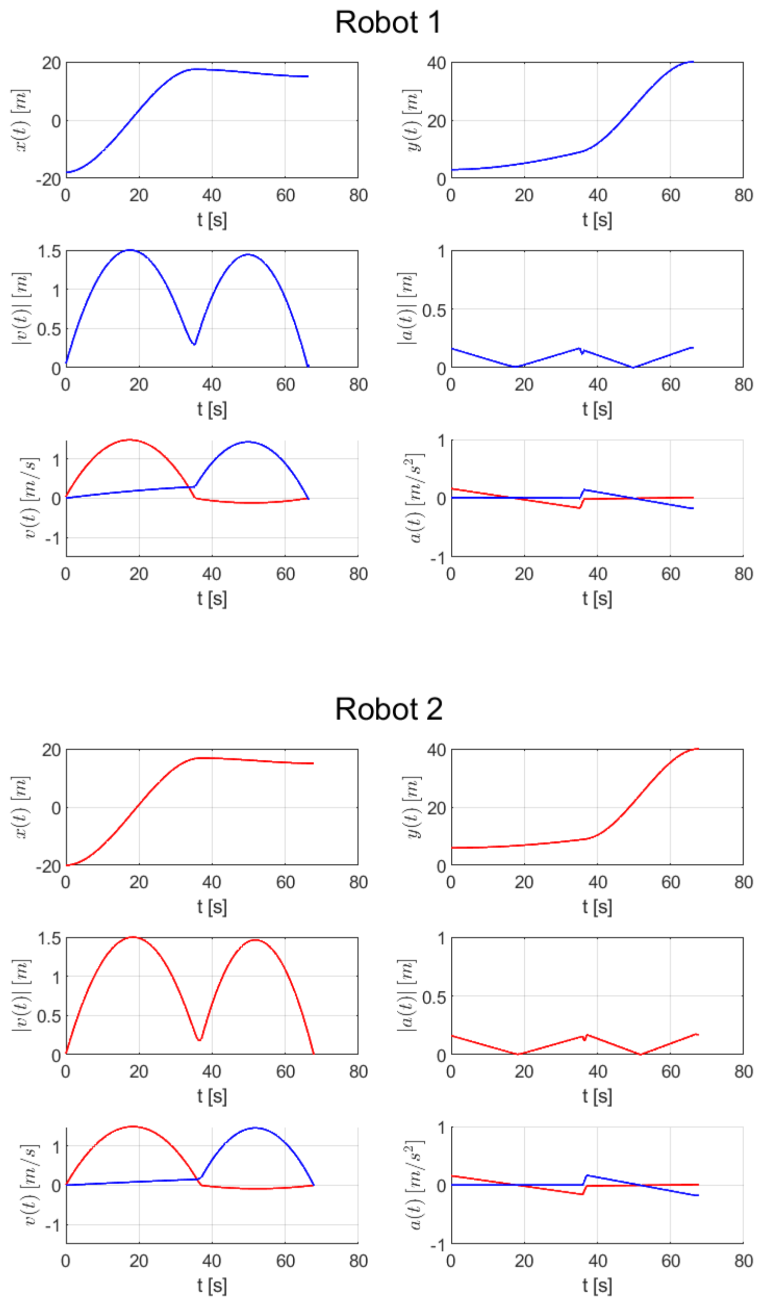


Figure 13: Position, velocity and acceleration profiles of Robot 1 and Robot 2, produced with the PCHIP algorithm starting from the kinematic constraints presented in table 1. The last two plots of both robots show the x (in red) and y (in blue) components of velocity and acceleration respectively.

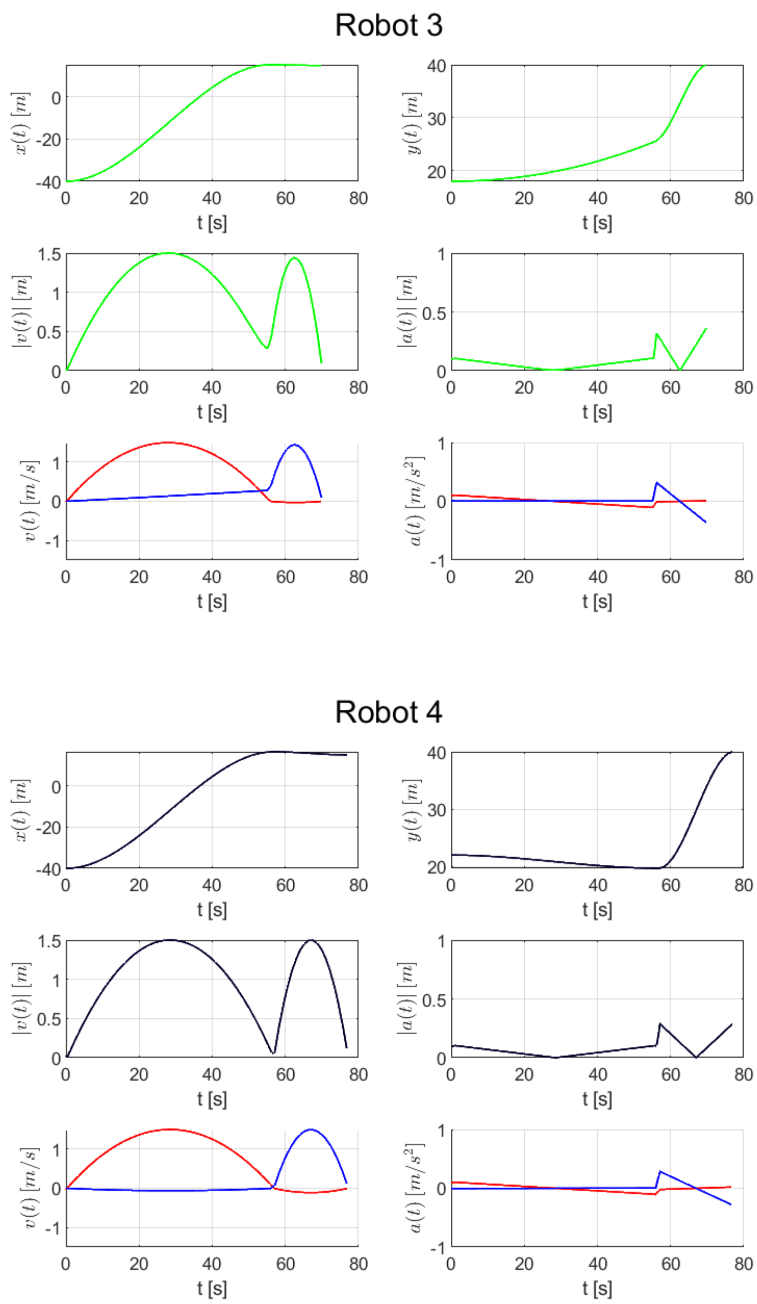


Figure 14: Position, velocity and acceleration profiles of Robot 3 and Robot 4, produced with the PCHIP algorithm starting from the kinematic constraints presented in table 1. The last two plots of both robots show the x (in red) and y (in blue) components of velocity and acceleration respectively.

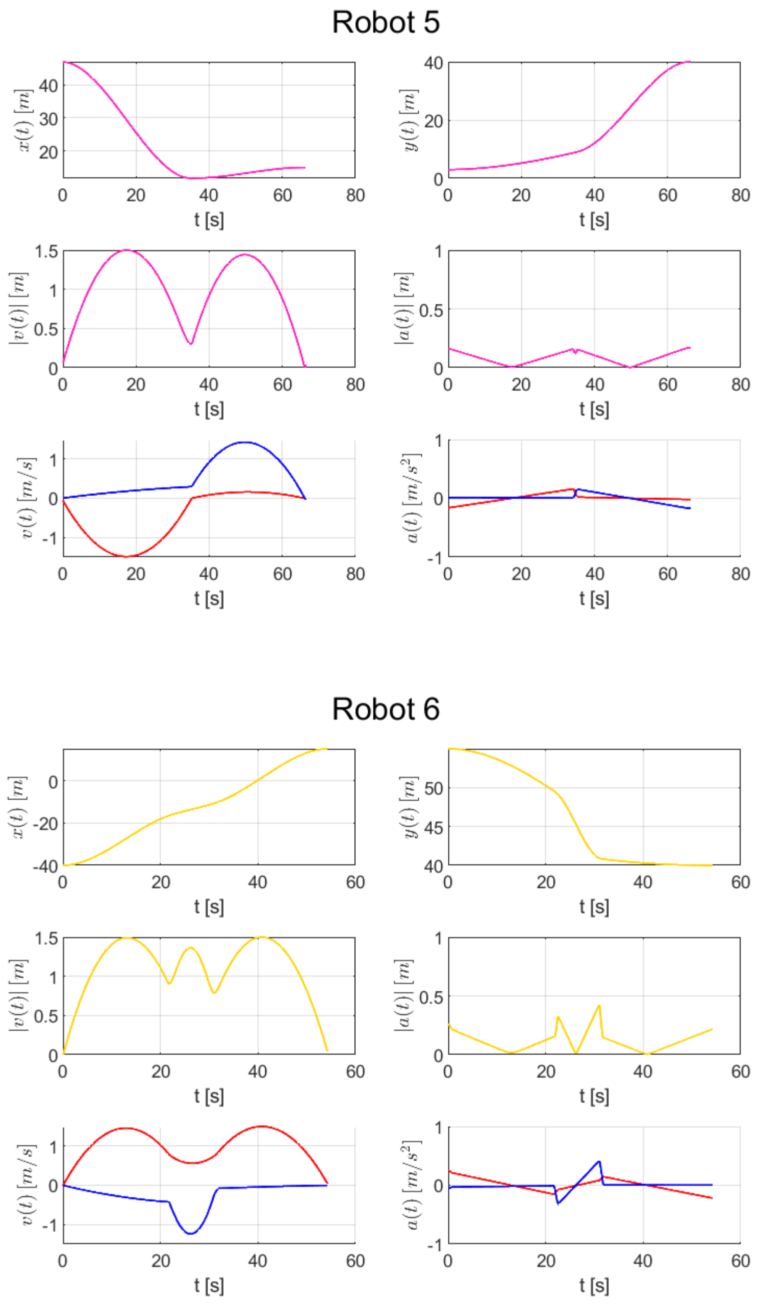


Figure 15: Position, velocity and acceleration profiles of Robot 5 and Robot 6, produced with the PCHIP algorithm starting from the kinematic constraints presented in table 1. The last two plots of both robots show the x (in red) and y (in blue) components of velocity and acceleration respectively.

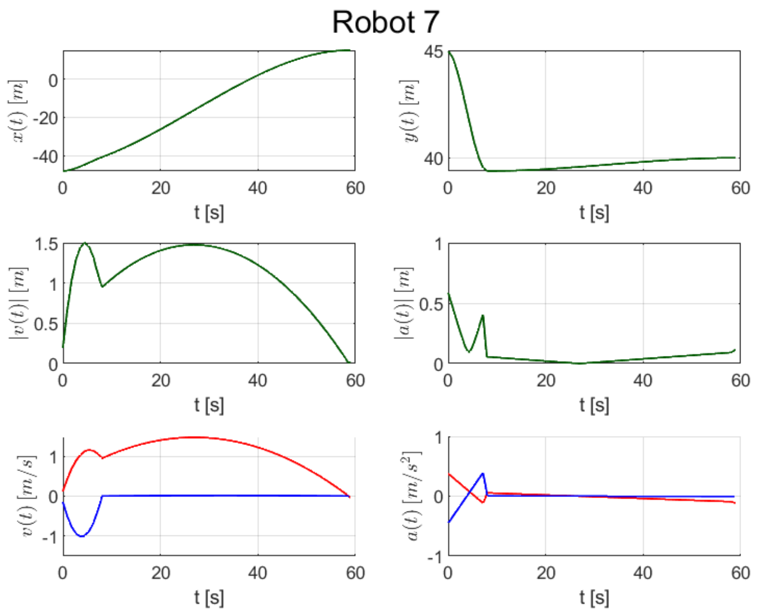


Figure 16: Position, velocity and acceleration profiles of Robot 7, produced with the PCHIP algorithm starting from the kinematic constraints presented in table 1. The last two plots of both robots show the x (in red) and y (in blue) components of velocity and acceleration respectively.

6.2 Evaluation criteria

The primary performance metrics evaluated in this study are total finish time, total energy spent and minimum distance between robots. The total finish time represents the duration required for the last robot to reach the goal, reflecting the efficiency of the system in completing the task. Total energy spent accounts for the energy spent by the system of all robots during their trajectories, which is critical for assessing the operational cost and sustainability of the system. The minimum distance between robots is a safety metric, ensuring that the trajectories maintain the right distance to avoid collisions, validating the effectiveness of the collision avoidance strategy. It is computed as the minimum distance between any pair of robots of the system at each time-step of the simulation:

$$d(t_k) = \min\Big\{ ||p_i(t_k) - p_j(t_k)|| : i, j = 1, ..., n \wedge i \neq j \wedge k \leq \min(N_i, N_j) \Big\} \tag{15}$$

Figure 23 shows the plot of $d(t_k)$ for the starting scenario of the simulations over all the time-steps, while table 2 shows the values of the total finish time and the total energy spent for the simulation.

Evaluation parameter	**Value**
Total finish time	76.80 s
Total energy spent	3.002 kJ

Table 2: Values for the evaluation parameters in the starting scenario.

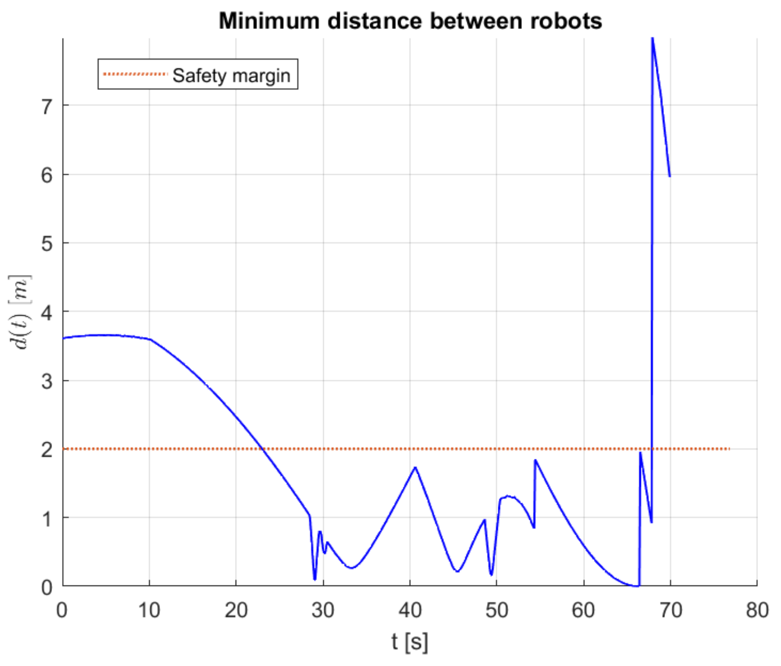


Figure 17: Plot of the minimum distance between any of the two robots of the system over time for the starting scenario.

6.3 Brute-force approach

The results of the brute force approach are obtained by evaluating all possible combinations of the maximum velocities of the robots along their respective trajectories, within predefined ranges. Specifically:

- In its first iteration the algorithm looks among all the 2^7 possible combinations of maximum velocities in the range $\left[v_{min}, v_{max}\right] = \left[1.4, 1.5\right]$ m/s, with a discretization step $d = 0.1$, so that only the combinations with 1.4 and 1.5 m/s are considered.
- At the second iteration, since no collision-free combination is found, the range is expanded to $\left[v_{min}, v_{max}\right] = \left[1.3, 1.5\right]$ m/s, so that at this stage there are 3^7 possible combinations to look for.
- After several iterations, the algorithm finds collision-free combinations in the range $\left[v_{min}, v_{max}\right] = \left[0.9, 1.5\right]$ m/s.

This means that the total number of combinations explored is:

$$(2^7 + 3^7 + ... + 7^7) \approx 1.2 \cdot 10^6 \tag{16}$$

This exponential growth in the number of possibilities highlights the computational intensity of the brute force method, especially for larger systems. Furthermore, the quality of the solution is directly influenced by the size of the step d: smaller steps allow for a finer resolution in the search space, leading to more precise results, but at the cost of significantly increasing the computational weight. This trade-off between accuracy and computational feasibility is a critical consideration when employing brute force optimization.

Once these combinations are found, the best combination that minimizes the criterion in (8) is selected. Table 3 summarizes the parameters and the results obtained with this approach.

Parameter	**Value**
d	0.1
Solution variable	**Value**
v^*	$\begin{bmatrix} 1.5 & 1.3 & 1.5 & 1.2 & 0.9 & 0.9 & 1.5 \end{bmatrix}$
Evaluation parameter	**Value**
Total finish time	110.90 s
Total energy spent	2.147 kJ
Execution time	84 min

Table 3: Outline of the parameters and the results obtained with the Brute-force approach.

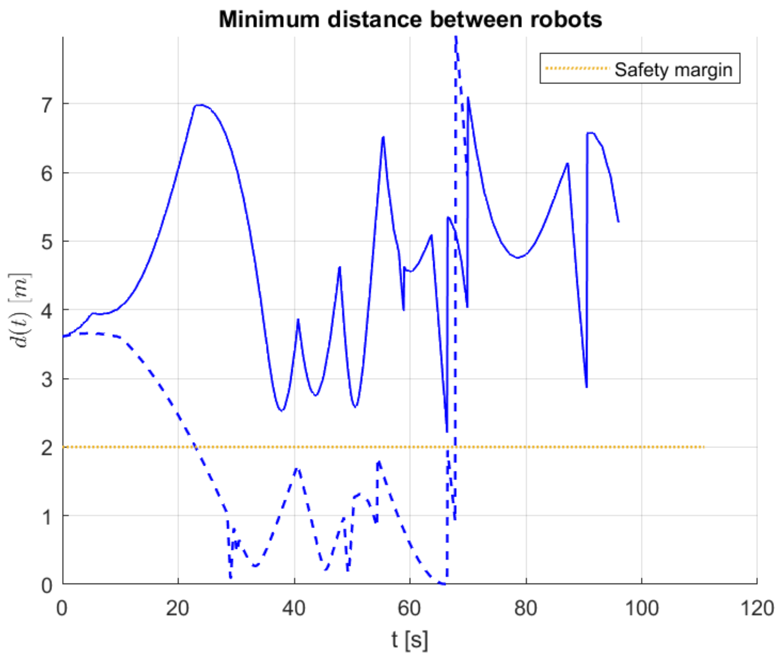


Figure 18: Plot of the minimum distance between any of the two robots of the system over time before (dashed line) and after (full line) applying the Brute-force algorithm.

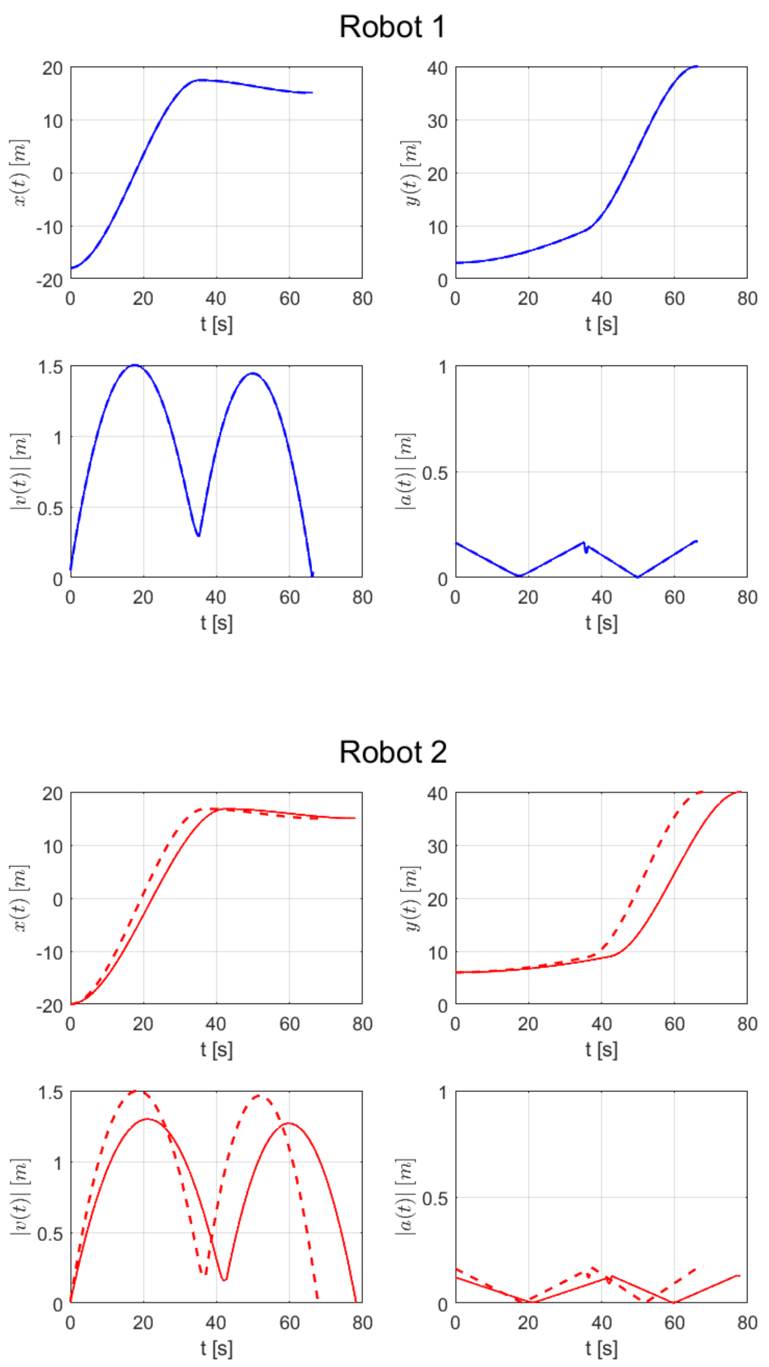
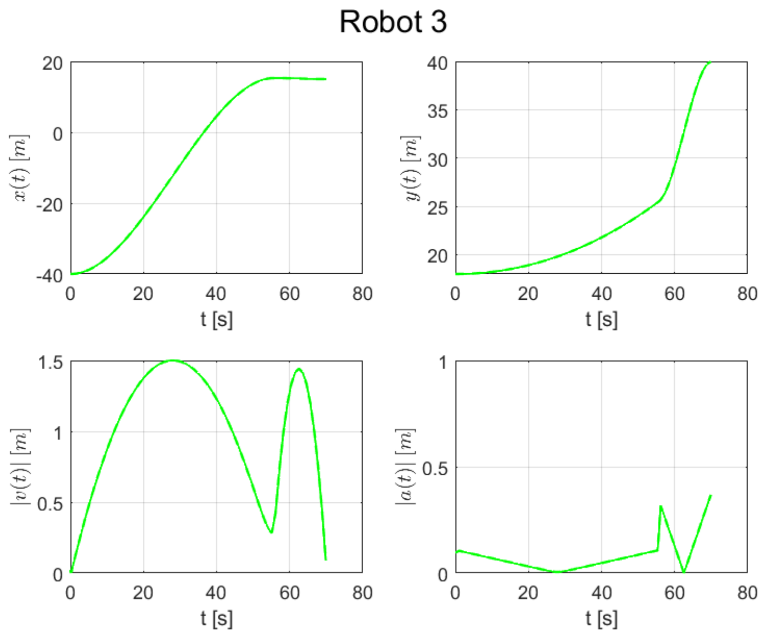


Figure 19: Position, velocity and acceleration profiles of Robot 1 and Robot 2, before (dashed line) and after (full line) applying the Brute-force algorithm.



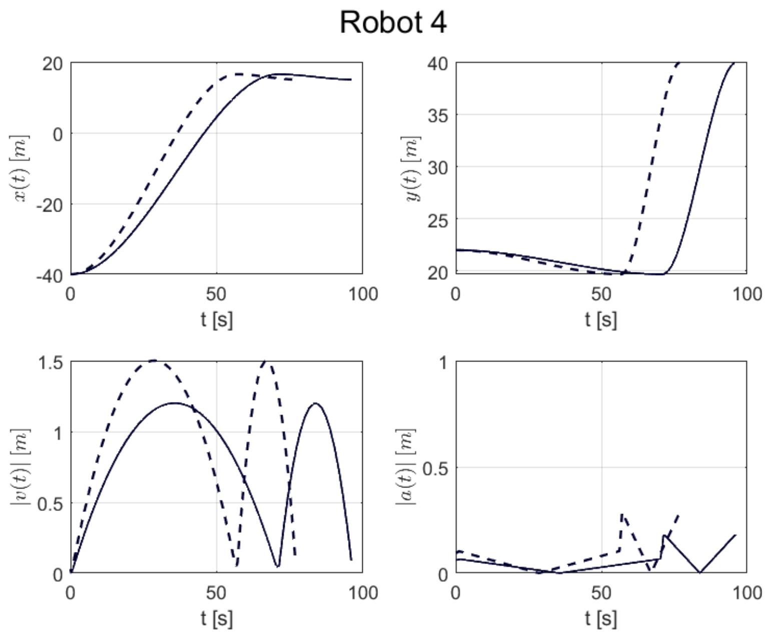
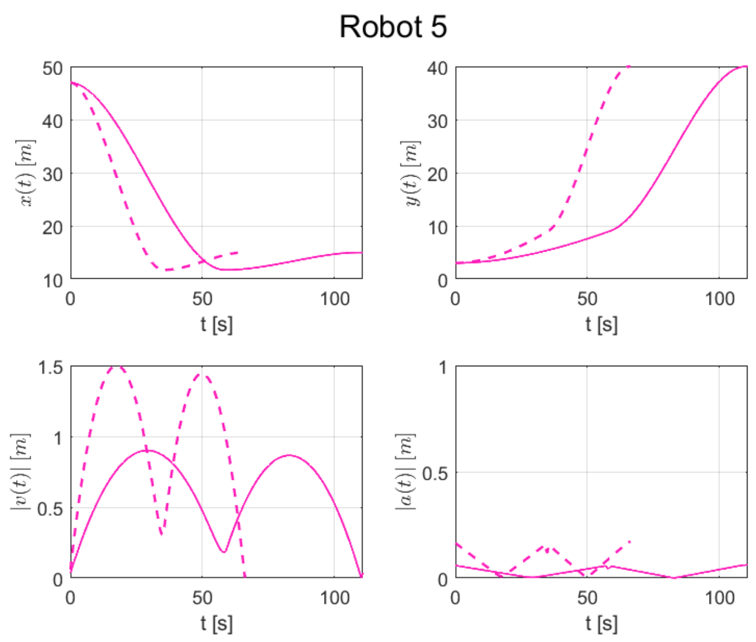


Figure 20: Position, velocity and acceleration profiles of Robot 3 and Robot 4, before (dashed line) and after (full line) applying the Brute-force algorithm.



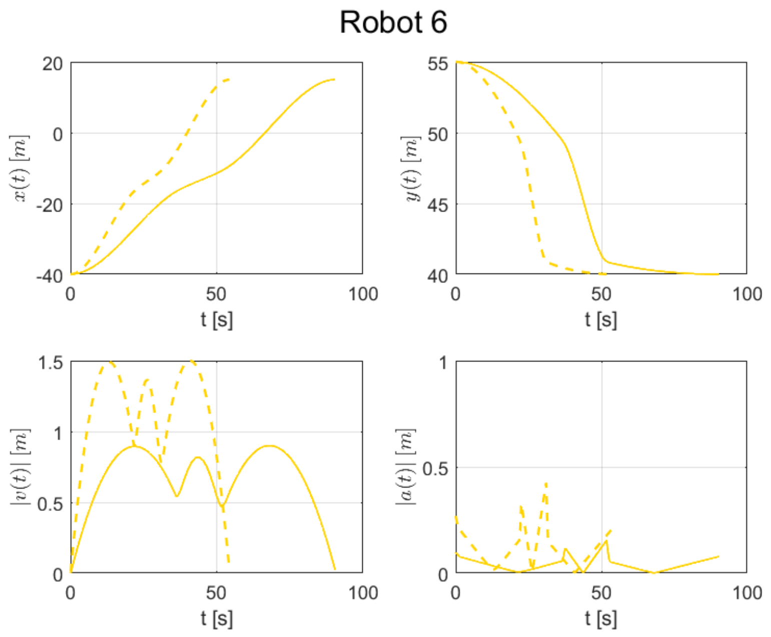


Figure 21: Position, velocity and acceleration profiles of Robot 5 and Robot 6, before (dashed line) and after (full line) applying the Brute-force algorithm.

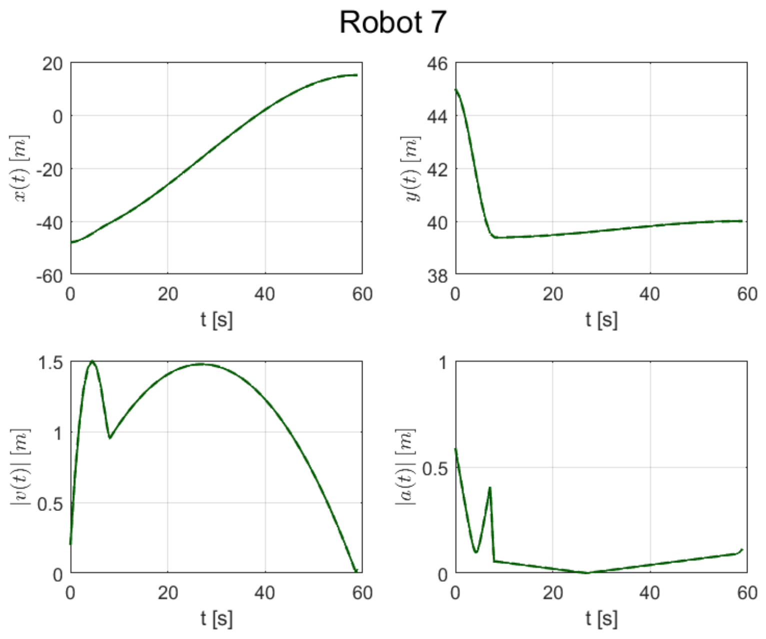


Figure 22: Position, velocity and acceleration profiles of Robot 7, before (dashed line) and after (full line) applying the Brute-force algorithm.

6.4 Prioritized planning

Since the Prioritized planning approach is significantly faster in terms of execution time than the previous approach, it is possible to choose a finer discretization step of 0.01. Simulations show that this approach results in a finish time of 126.30 seconds, which is worse than the 110.90 seconds achieved with the brute-force method. This difference highlights the trade-off in the prioritized planning approach, where execution time is prioritized over the quality of the result.

Parameter	**Value**
d	0.01
Solution variable	**Value**
v^*	$\begin{bmatrix} 1.03 & 1.50 & 1.20 & 1.50 & 0.79 & 0.77 & 1.50 \end{bmatrix}$
Evaluation parameter	**Value**
Total finish time	126.30 s
Total energy spent	1.957 kJ
Execution time	0.44 s

Table 4: Outline of the parameters and the results obtained with the Prioritized planning approach.

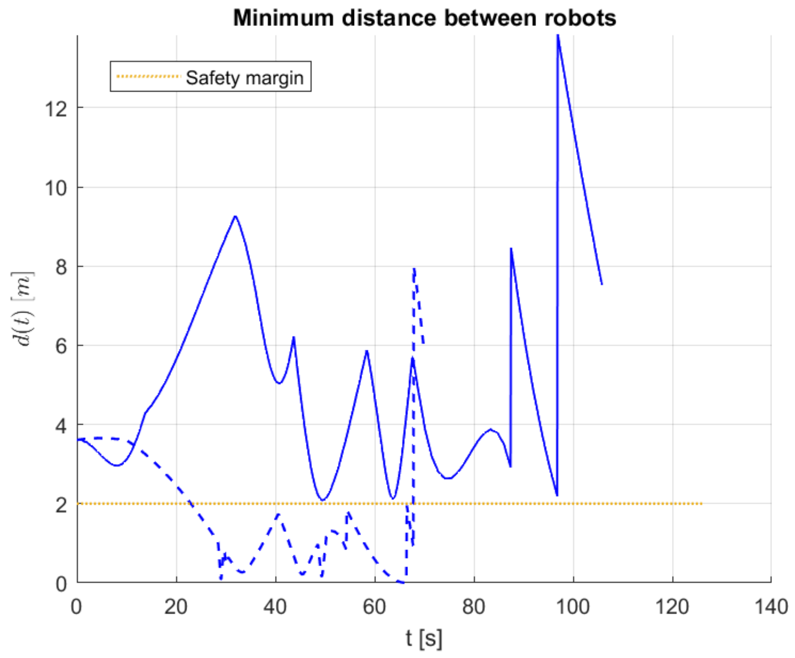


Figure 23: Plot of the minimum distance between any of the two robots of the system over time before (dashed line) and after (full line) applying the Prioritized planning algorithm.

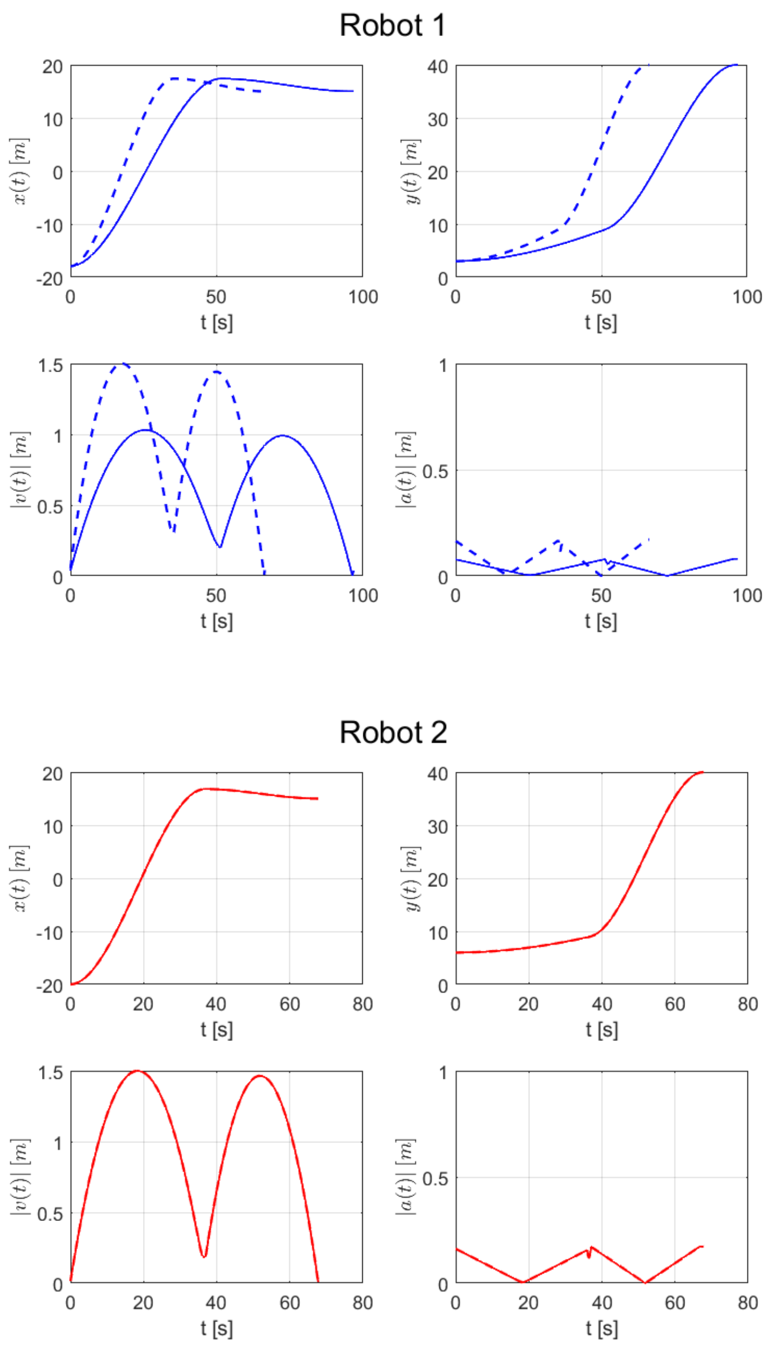
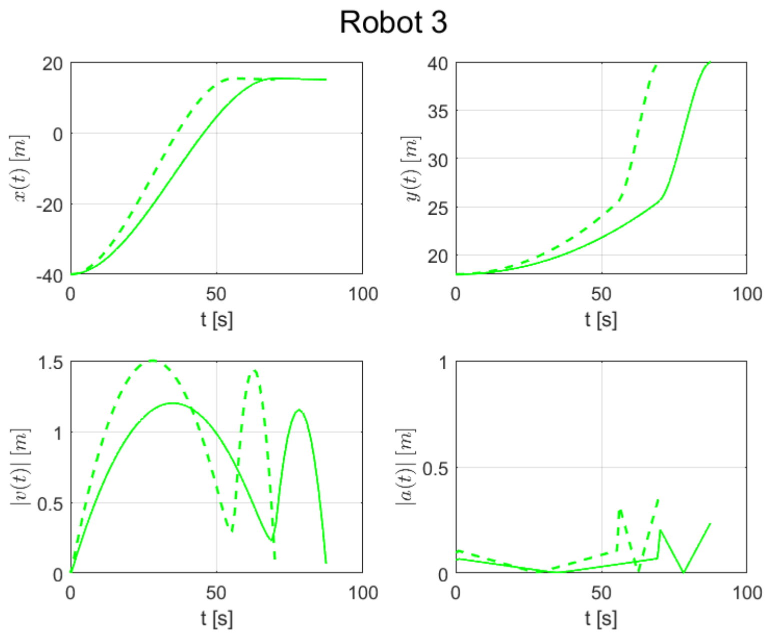


Figure 24: Position, velocity and acceleration profiles of Robot 1 and Robot 2, before (dashed line) and after (full line) applying the Prioritized planning algorithm.



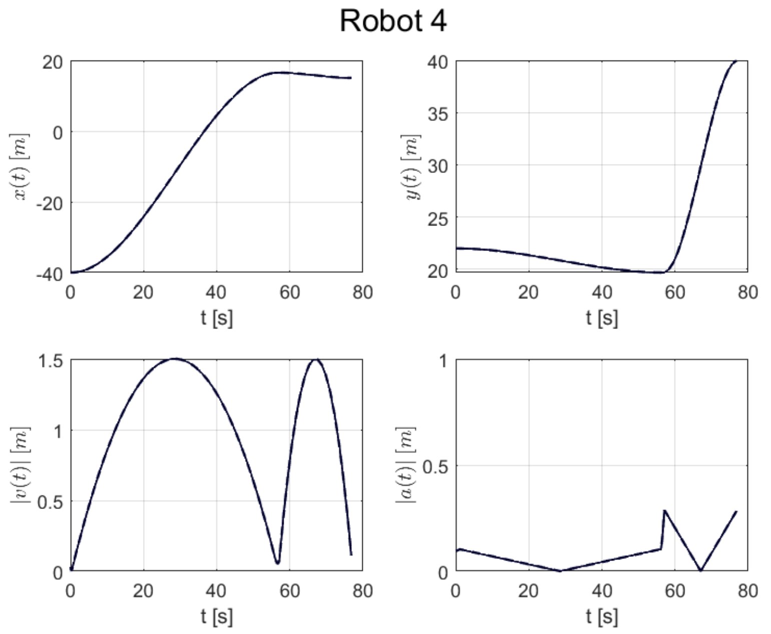
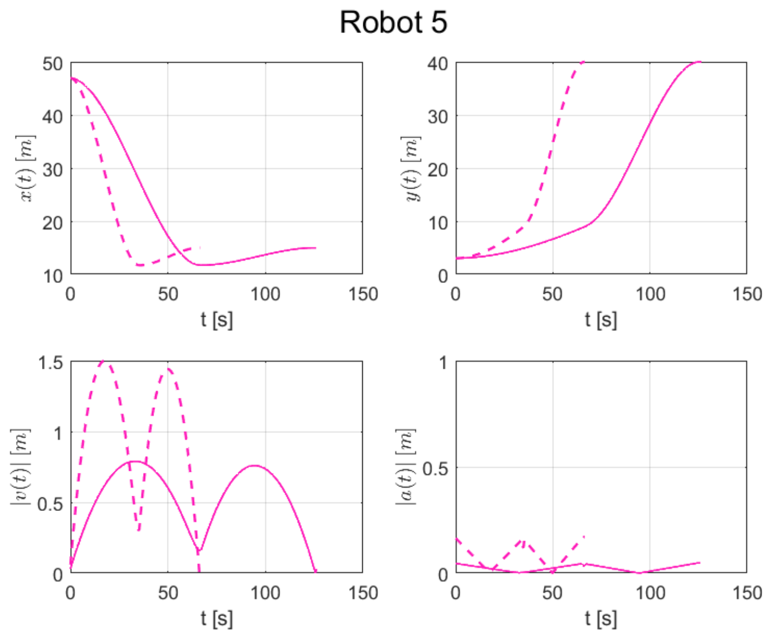


Figure 25: Position, velocity and acceleration profiles of Robot 3 and Robot 4, before (dashed line) and after (full line) applying the Prioritized planning algorithm.



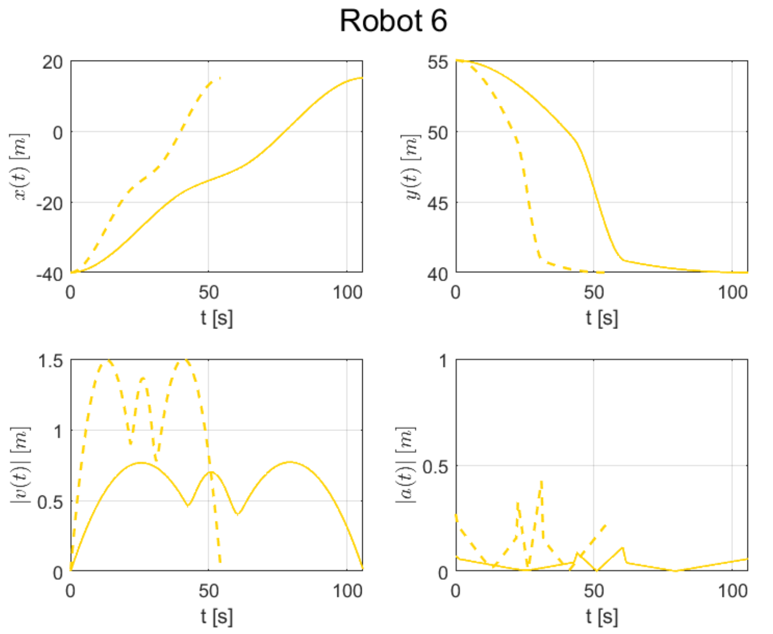


Figure 26: Position, velocity and acceleration profiles of Robot 5 and Robot 6, before (dashed line) and after (full line) applying the Prioritized planning algorithm.

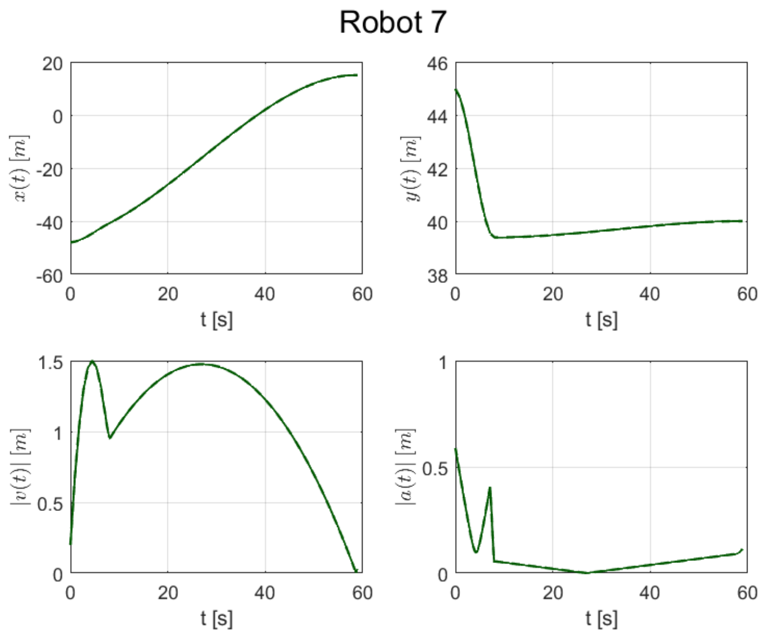


Figure 27: Position, velocity and acceleration profiles of Robot 7, before (dashed line) and after (full line) applying the Prioritized planning algorithm.

6.5 Maximum velocities optimization

In this chapter, I conduct various simulations using the Maximum velocities optimization approach, introduced in section 5.2, using the Genetic Algorithm. To analyze the performance and robustness of this approach, I vary key parameters such as the Crossover Fraction and Mutation Rate. Laplace crossover and Uniform mutation, introduced in section 2.5, are used for all the simulations.

6.5.1 Constraints analysis

To evaluate the impact of Mutation Rate and Crossover Fraction on the algorithm's performance, 50 independent simulations are conducted across 9 different parameter combinations. Each simulation is executed with a Population Size of 100 and ran for 500 Generations. A critical aspect of this evaluation involved analyzing the constraints, as satisfying these constraints is fundamental to ensuring the feasibility of the solutions. Constraints define the boundaries within which the optimization operates, and their violation indicates a failure to generate valid solutions that meet the problem's requirements.

The analysis of the constraints, shown in Figure 28, reveals that the majority of simulations fails to satisfy them, as indicated by constraint function values exceeding zero in most cases.

To provide further insights, Figure 29 illustrates the number of successful executions for each parameter combination. A successful execution is defined as a simulation where all constraints are satisfied by the end of the 500 generations. The results demonstrate that the method struggles to consistently identify feasible solutions, highlighting the need for refinements in the algorithm to improve its ability to meet the imposed constraints effectively. This analysis is crucial for identifying weaknesses in the optimization approach and guiding future improvements.

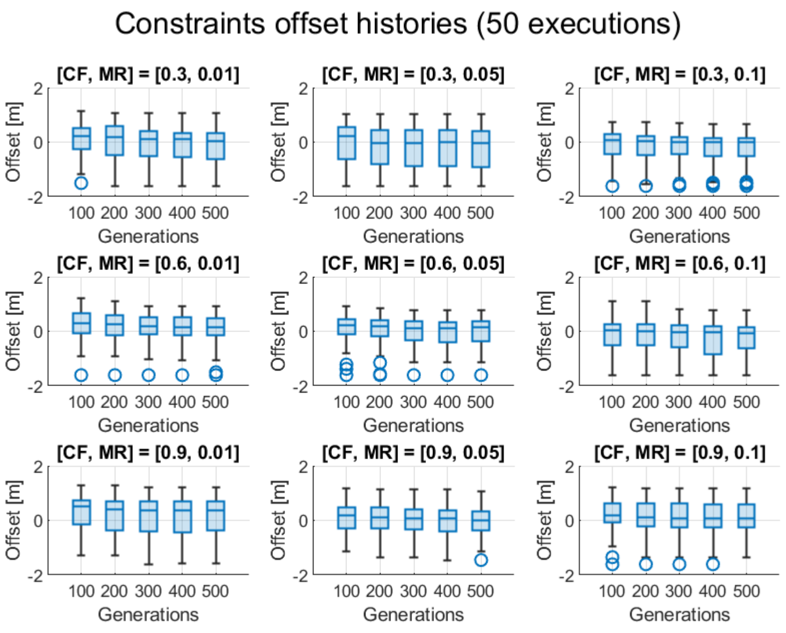


Figure 28: Box charts of 50 simulations executed with 9 different combinations of Mutation Rate and Crossover Fraction, where every simulation is performed with a population size of 100 individuals. The plots show the distributions of the values of the constraint function for every simulation.

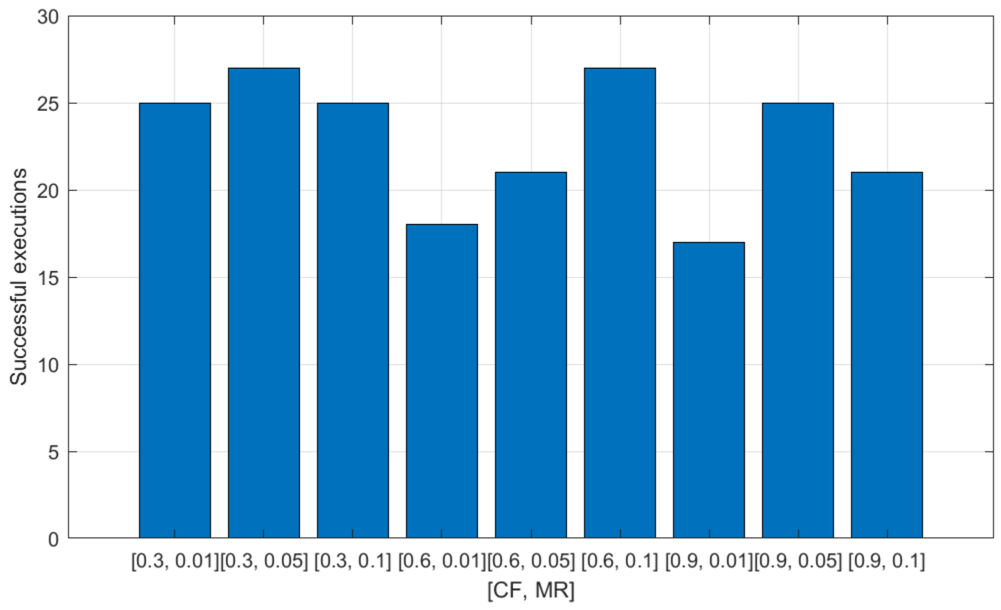


Figure 29: Bar chart for the successful executions after 500 generations, over 50 executions for 9 combinations of Mutation Rate and Crossover Fraction, with a Population Size of 100.

6.6 Slow-down segment optimization

The previous method, which focused on optimizing the maximum velocities of the robots, struggled in finding feasible solutions, with the majority of simulations failing to meet the imposed constraints. To address this limitation, a more effective approach involves the Slow-down segment optimization, described in section 5.3, which introduces flexibility by allowing specific segments of the robots' paths to be adjusted to avoid collisions.

This section presents the results of applying the Slow-down segment optimization for the same combinations of Mutation Rate and Crossover Fraction used in the Maximum velocities optimization in the previous section. The simulations show an improved ability to find feasible solutions, highlighting the potential of this approach to better handle the problem's constraints and achieve effective collision avoidance. These simulations also aim to explore the effectiveness of the proposed method under different configurations and provide insights into the optimal parameter settings, through the analysis performed in section 6.6.2.

6.6.1 Effect on the trajectories

First, the effects of the Slow-down segment optimization on the trajectories of the robots are presented. The map shown in 30 illustrates the optimized slow-down segments on the trajectories of the robots. These segments represent the areas where velocity reduction is implemented to avoid collisions while optimizing the travel time. Additionally, the subsequent figures provide a detailed analysis of the velocity profiles of the robots, highlighting how the slow-down segments influence their motion.

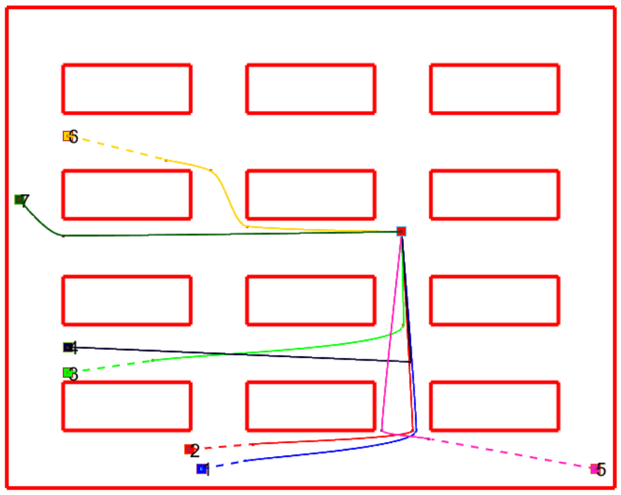


Figure 30: Map and trajectories modified with the slow-down segment computed through the Slow-down segment optimization algorithm.

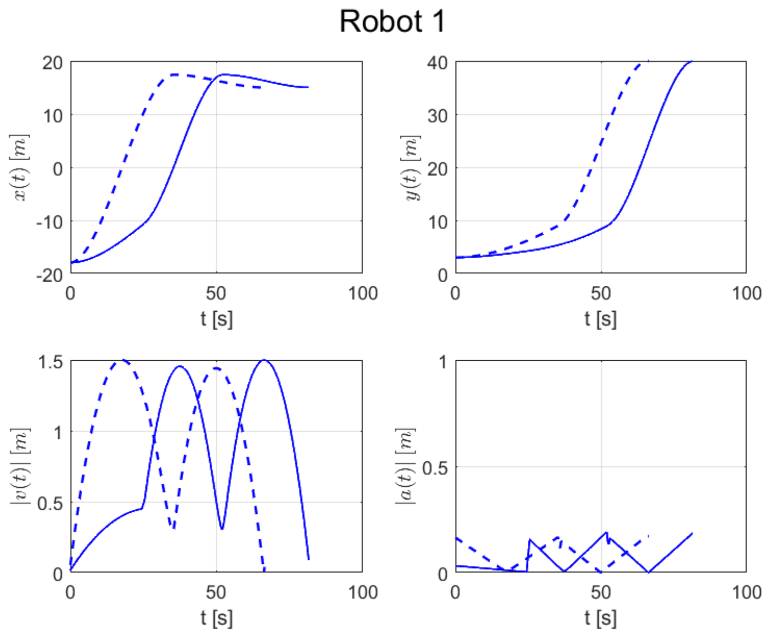
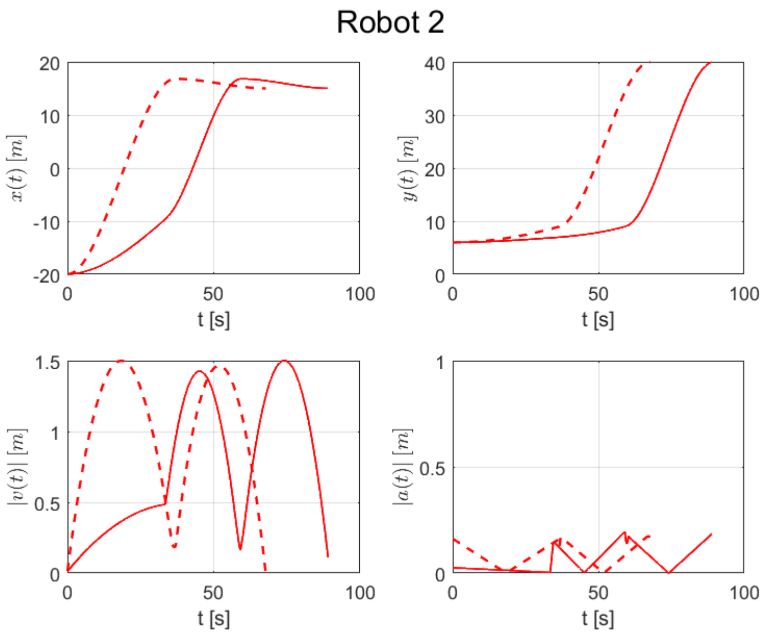


Figure 31: Position, velocity and acceleration profiles of Robot 1, before (dashed line) and after (full line) applying the Slow-down segment optimization algorithm.



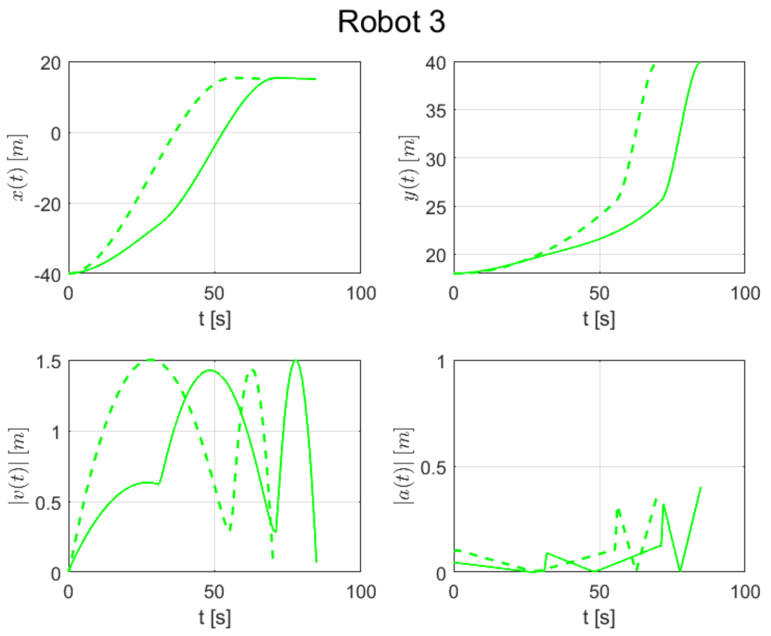
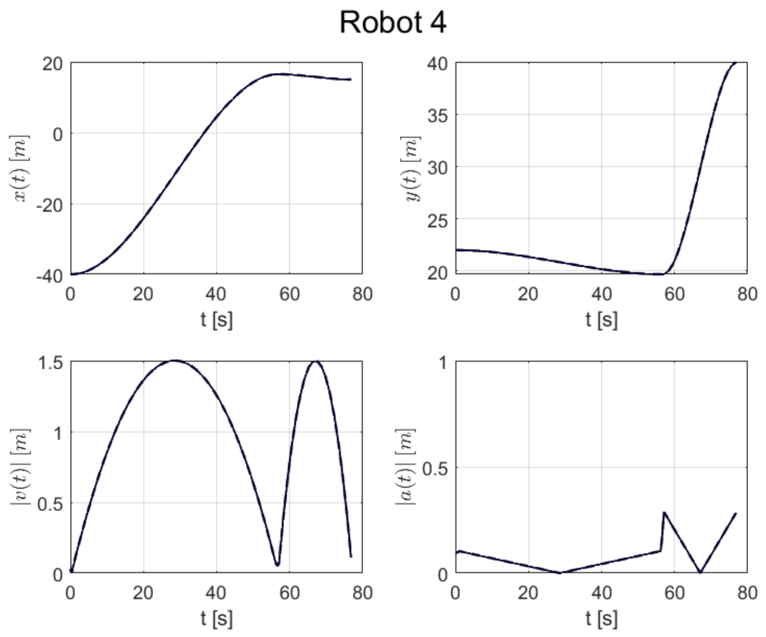


Figure 32: Position, velocity and acceleration profiles of Robot 2 and Robot 3, before (dashed line) and after (full line) applying the Slow-down segment optimization algorithm.



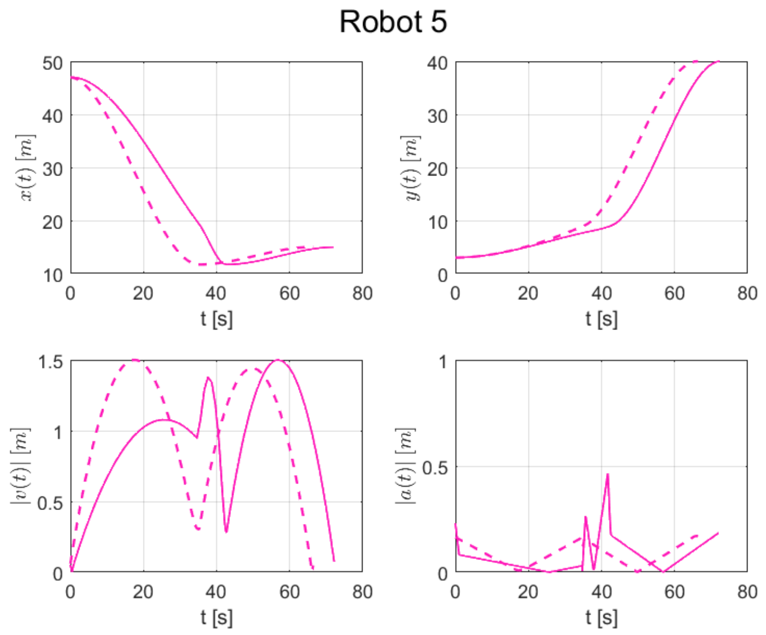
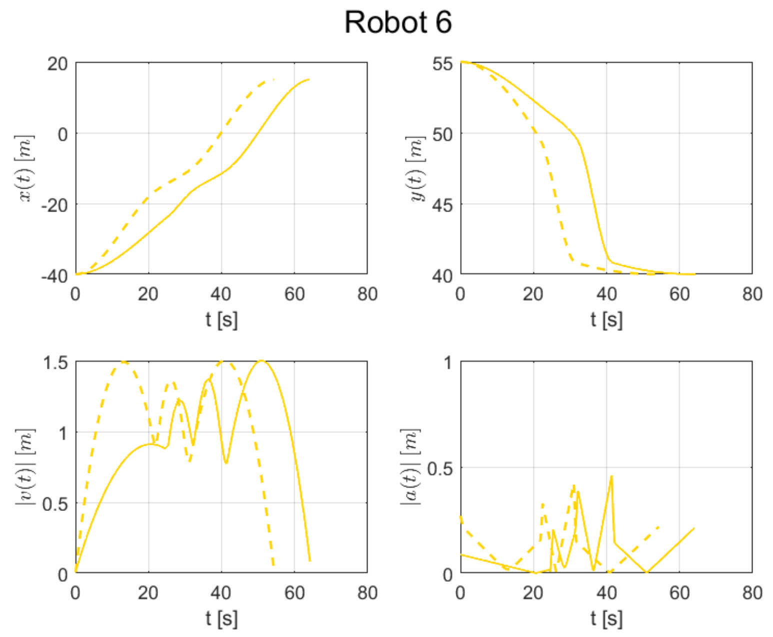


Figure 33: Position, velocity and acceleration profiles of Robot 4 and Robot 5, before (dashed line) and after (full line) applying the Slow-down segment optimization algorithm.



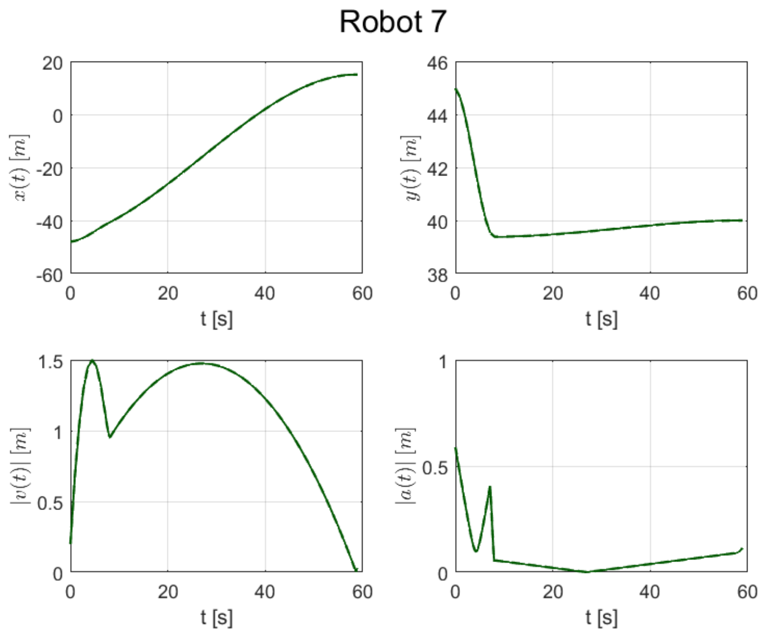


Figure 34: Position, velocity and acceleration profiles of Robot 6 and Robot 7, before (dashed line) and after (full line) applying the Slow-down segment optimization algorithm.

6.6.2 Parameters selection

A first batch of simulations are conducted to analyze the impact of Mutation rate and Crossover fraction on the performance of the algorithm. As in section 6.5, a total of 50 independent executions are performed for 9 combinations of these parameters, with each execution consisting of 500 Generations and a Population size of 100, to analyze their impact on the algorithm's performance.

The constraints are analyzed first to evaluate the validity of the solutions generated by the algorithm. Once again, not in all simulations the constraints are satisfied, as evidenced in figure 35 by some values of the constraint function exceeding zero. This indicates that certain combinations of parameters result in solutions that violate the feasibility requirements. Results show that the combinations with the lowest Crossover Fraction produce better results, as the distribution of the values of the constraint function settles to the lowest values, highlighting the importance of a broader exploration favored by a higher amount of muted solutions.

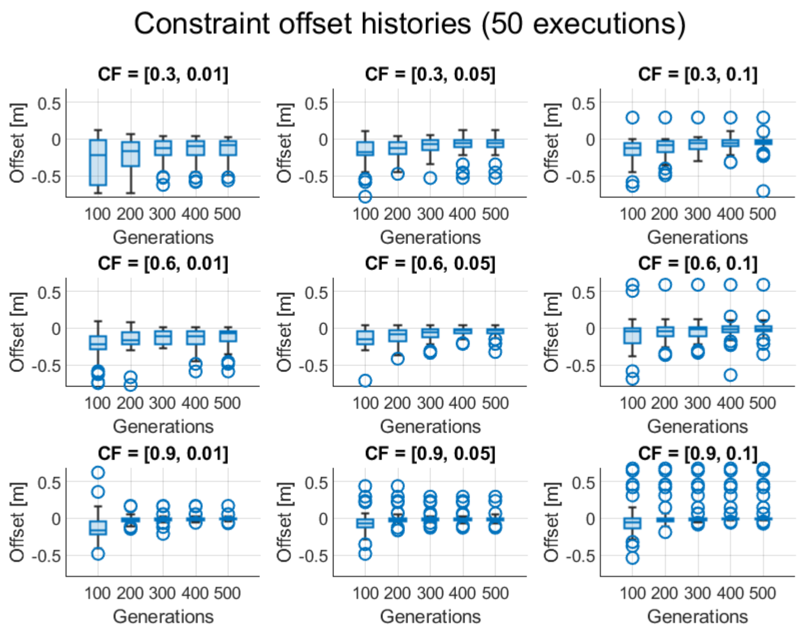


Figure 35: Box charts of 50 simulations executed with 9 different combinations of Mutation rate and Crossover fraction, where every simulation is performed with a population size of 100 individuals.

To ensure a meaningful analysis of the algorithm's performance, only simulations where the constraints are respected from the very first generation are considered for further evaluation. This filtering step is crucial to ensure that the results accurately reflect the true capability of the algorithm, without interference from violations that could distort the analysis.

The decision to exclude simulations with constraint violations is due to the way the fitness function is formulated. Specifically, the fitness function incorporates Lagrangian multipliers, which adjust the fitness value when constraints are not respected. These adjustments penalize solutions that violate the constraints, resulting in skewed fitness values that do not accurately represent the algorithm's actual optimization performance. This approach provides a clearer and more reliable comparison of the algorithm's effectiveness across different parameter combinations.

For the subset of valid simulations, the best fit values are plotted across generations to investigate the convergence behavior of the algorithm. The evolution of the best fit values in figure 36 reveal patterns that vary significantly with different parameter combinations, highlighting the influence of the Crossover Fraction and Mutation Rate on the algorithm's ability to explore and exploit the solution space effectively.

In particular, the analysis reveals the influence of the Mutation Rate on the algorithm's convergence behavior and solution quality. Higher values of the Mutation Rate are observed to facilitate the algorithm's ability to escape local optima and explore a broader range of the solution space. This is particularly evident when the Mutation Rate is increased from 0.01 to 0.1, resulting in noticeably improved final best-fit values. The increased randomness introduced by a higher Mutation Rate helps the algorithm avoid premature convergence by introducing new genetic material into the population. This diversification ensures that the search process does not stop and continues to identify better solutions as the generations progress.

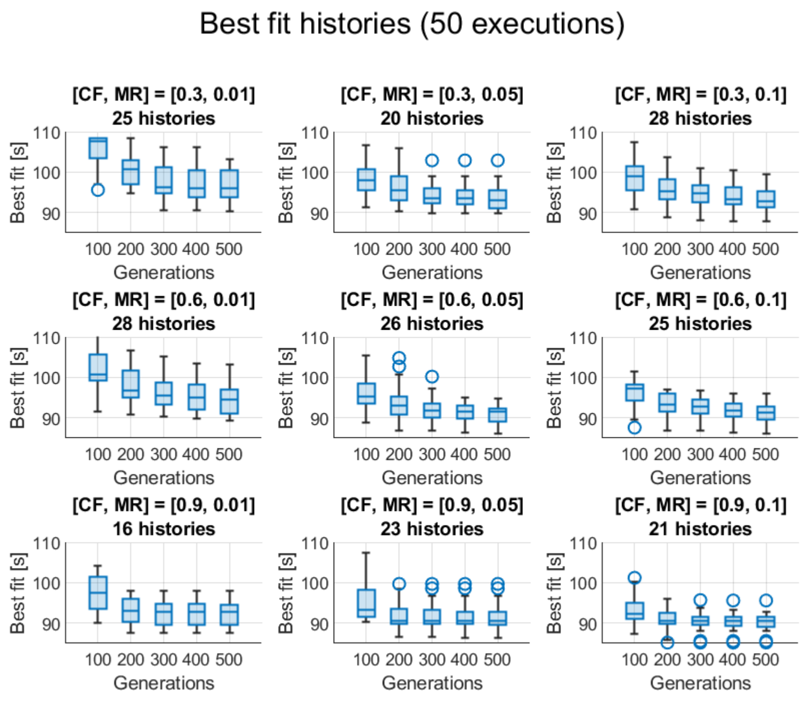


Figure 36: Box charts of 50 simulations executed with 9 different combinations of Mutation rate and Crossover fraction, where every simulation is performed with a Population Size of 100 individuals. Only the simulations in which the constraints are respected for all the generations are included in the data.

To further investigate the impact of the Mutation Rate and Crossover Fraction, I also analyze the subset of simulations where the constraints are satisfied at the end of the generations, referred to as successful executions. A successful execution is defined as a simulation in which all constraint values remain within acceptable bounds (i.e., less than or equal to zero) by the final generation. For these simulations, I plot the final best-fit values, which reveal that higher Mutation Rates contribute to better convergence, even though the resulting medians are quite similar across different Mutation Rates, as shown in figure 38.

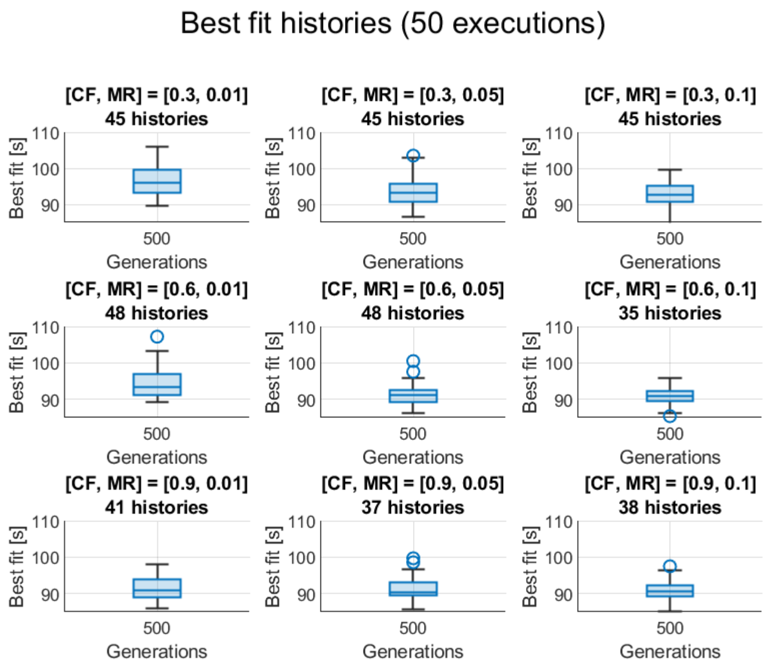


Figure 37: Box charts of 50 simulations executed with 9 different combinations of Mutation rate and Crossover fraction, where every simulation is performed with a Population Size of 100 individuals. Only the simulations where the constraints are respected at the end of all the generations are included in the data.

Additionally, I plot the number of successful executions for each parameter combination in figure 38. Results show that lower Crossover Fractions produce a higher number of successful executions. This can be attributed to the increased exploratory behavior of the algorithm when the Crossover Fraction is reduced, allowing the population to explore a wider variety of potential solutions. Although this comes at the cost of slower convergence, it demonstrates the trade-off between exploration and exploitation in the optimization process. These results, compared to those in figure 29, show that this algorithm is more effective in finding feasible solutions than the Maximum velocities optimization algorithm.

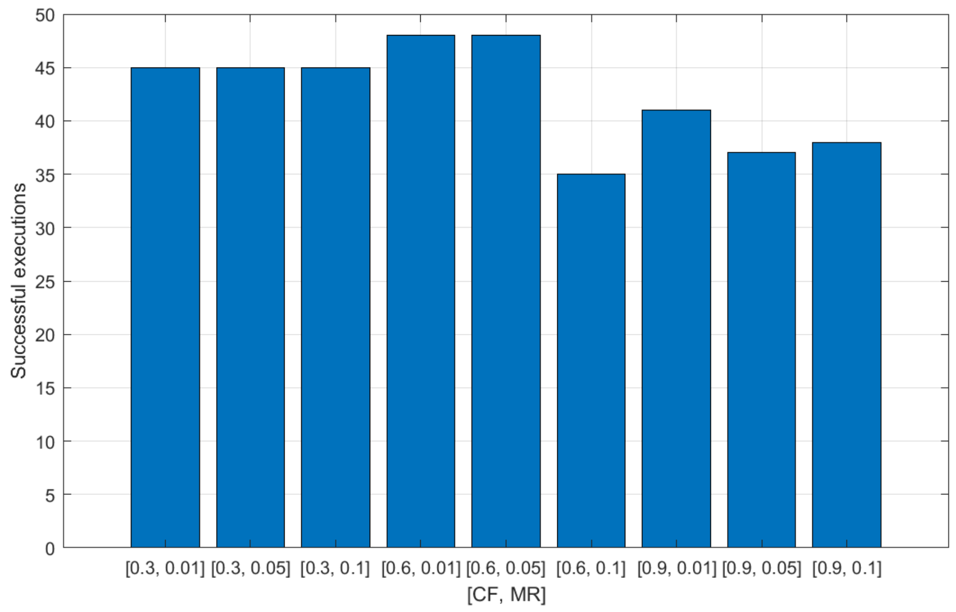


Figure 38: Bar chart for the number of successful simulations over 50 executions, for different combinations of Mutation Rate and Crossover Fraction.

6.7 Finish time and energy consumption optimization

6.8 Comparison of the approaches

7 Conclusions and future work

References

[1] Gonçalo Teixeira. *Fleet Assembly Optimisation for Cooperative Ground Transporters In Constrained Environments.* 2024.

[2] Foued Saadaoui Rafaa Mraihi Bochra Rabbouch, Hana Rabbouch. *Comprehensive Metaheuristics: Algorithms and Applications, Chapter 22: Foundations of combinatorial optimization, heuristics, and metaheuristics.* Academic Impress, Elsevier, 2023.

[3] Sven Leyffer Jeff Linderoth Jim Luedtke Pietro Belotti, Christian Kirches and Ashutosh Mahajan. *Mixed-Integer Nonlinear Optimization.* Acta Numerica, Volume 22, 2013.

[4] Amit Dixit Mohamed Benaida Tanweer Alam, Shamimul Qamar. *Genetic Algorithm: Reviews, Implementations, and Applications.* International Journal of Engineering Pedagogy (iJEP), 2020.

[5] Jianguo Wang Xiaoying Kong Shiwei Lin, Ang Liu. *A Review of Path-Planning Approaches for Multiple Mobile Robots.* Machines, 2022.

[6] Dinesh Manocha Jur van den, Berg Ming Lin. *Reciprocal Velocity Obstacles for Real-Time Multi-Agent Navigation.* Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2008.

[7] J.E. Smith A.E. Eben. *Introduction to evolutionary computing, Chapter 4: Representation, Mutation and Recombination.* Springer, 2015.

[8] Krishna Pratap Singh Chander Mohan Kusum Deep, M.L. Kansal. *A real coded genetic algorithm for solving integer and mixed integer optimization problems.* Applied Mathematics and Computation, Elsevier, 2009.

[9] Mathworks. Genetic algorithm options - crossover options. `https://www.mathworks.com/help/gads/genetic-algorithm-options.html`. [Online; accessed 15-November-2024].

[10] Michele Lombardi Allegra De Filippo and Michela Milano. *Off-Line and On-Line Optimization Under Uncertainty: A Case Study on Energy Management.* Integration of Constraint Programming, Artificial Intelligence, and Operations Research. 15th International Conference, CPAIOR, 2018.

[11] Miryam Pirozzi. *Miryam's paper.* 2024.

[12] Luigi Villani Giuseppe Oriolo Bruno Siciliano, Lorenzo Sciavicco. *Robotics: Modelling, planning and control.* Advanced Textbooks in Control and Signal Processing, 1998.

[13] Cleve B. Moler. *Numerical Computing with Matlab.* Society for Industrial and Applied Mathematics, 2004.

[14] Jur P. Van Den Berg. *Prioritized motion planning for multiple robots.* 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2005.

[15] Nicholas Moehle. *Optimal Racing of an Energy-Limited Vehicle.* 2013.

[16] Victor Fachinotti Facundo Bre. *A computational multi-objective optimization method to improve energy efficiency and thermal comfort in dwellings.* Energy and Buildings, 2017.