TÉCNICO LISBOA

Decentralized trajectory optimization for a fleet of industrial mobile robots

Inês Sofia Baptista Silva

Dissertation to obtain the Master of Science Degree in

Aerospace Engineering

Supervisor(s): Prof. Alberto Manuel Vale
Prof. Rodrigo Martins de Matos Ventura

Examination Committee

November 2022

Dedicated to my family and friends.

Acknowledgments

My first word of appreciation goes to my supervisors, Professor Rodrigo Ventura and Professor Alberto Vale, for their insightful suggestions and encouragement. Without them, the conclusion and success of this work would not be possible.

Thanks to my colleague and co-worker, João Tavares, for the teamwork and companionship. I would also want to thank João Mendes for providing an amazing working place at Imeguisa.

On a more personal level, a very special note goes to my brother, Luís, my mother, Fernanda and my father, Luís. Thank you for all the opportunities, unconditional love, and support you have provided me over the past 23 years.

I owe a big thanks to Carolina, Constança, Diogo and Vitória for teaching me the meaning of true friendship.

To Beatriz, Iara, Inês Carriço, Inês Fernandes, Rita Fardilha and Rita Santos goes all my gratitude for contributing to my success in IST.

Last but not least, a special thanks to João, for all his patience and love.

Resumo

O planeamento de trajectórias de vários robôs é uma tarefa complexa, dada a necessidade de evitar colisões com obstáculos estáticos e dinâmicos, tais como outros robôs em movimento. Esta dissertação apresenta uma nova abordagem para gerar sequencialmente trajectórias optimizadas e sem colisões. O método proposto está divido em duas fases. Na primeira, é gerada uma trajetória com base num algoritmo aleatório. De seguida, é definido um problema de otimização que suaviza a trajetória obtida anteriormente. Como principal contribuição desta dissertação, foram utilizados campos de *signed distance* para descrever os obstáculos dinâmicos ao longo do tempo. Assim, o problema de optimização formulado apresenta apenas uma restrição relativamente ao desvio de obstáculos. Deste modo, a complexidade do método desenvolvido é independente do número de agentes.

O principal objectivo desta dissertação é gerar descentralizadamente trajectórias optimizadas para uma frota de robôs móveis autónomos, que irão operar num ambiente industrial de fábrica. Deste modo, requisitos adicionais foram tidos em consideração. Primeiramente, o problema de optimização foi reformulado, para que a deslocação dos robôs seja realizada, preferencialmente, do lado direito dos corredores da fábrica. Além disso, será apresentado um algoritmo que garante que todos os veículos tenham um mapa actualizado do ambiente da fábrica. Esta atualizção é necessária sempre que outros detectem o aparecimento ou desaparecimento de obstáculos significativos. Foram efectuados vários testes em simulação, assim como testes experimentais num cenário real. No geral, o método proposto providenciou uma solução satisfatória de planeamento de trajectórias de vários robôs.

Palavras-chave: Sistemas com Múltiplos Robôs, Planemento de Movimento, Optimização de Trajetória, Desvio de Obstáculos, Atualização de Mapa Dinâmimco

Abstract

Multi-robot trajectory planning is a complex task given the need to avoid collisions with both static and dynamic obstacles, such as other moving robots. This dissertation presents a novel approach to sequentially generating collision-free optimized trajectories for multiple mobile robots. The proposed method first uses a sampling-based kinodynamic trajectory planner to obtain a collision-free trajectory. Then, a non-linear direct collocation method refines the previous trajectory into a smoother and optimized one. As the main novelty of this dissertation, a set of signed distance fields were used to describe the environment and the dynamic obstacles through time. Thus, this formulation presents only one restriction in the optimization problem regarding static and moving obstacles avoidance. This renders the solver's performance independent of the number of agents.

This dissertation's main purpose is to implement a decentralized trajectory optimization solution for a fleet of autonomous mobile robots, that will operate in an industrial manufacturing unit. Some additional requirements had to be considered for this application. A novel reformulation of the optimization problem was designed in order to ensure that the robots drive on the right side of the factory's corridors. Additionally, an algorithm will be reported that allows all vehicles to have an up-to-date map of their surroundings, whenever other robots detect changes in it. Several simulation tests, as well as experimental ones in a real-world scenario, were carried out. Overall, the proposed framework provided a satisfactory solution to the multi-robot planning problem.

Keywords: Multi-Robot Systems, Motion planning, Trajectory Optimization, Collision Avoidance, Dynamic Map Update

Contents

List of Tables

List of Figures

Nomenclature

β	angular range of the vehicle sensor
$\mathcal{T}$	Rapidly-exploring Random Tree tree
$\mathbf{f}$	system dynamics
$\mathbf{p}$	pose of a robot, $\mathbf{p}(t) in \mathbf{x}(t)$
$\mathbf{u}$	input control of the vehicle, $u \in \mathcal{U}$
$\mathbf{x}$	state of the vehicle, $x \in \mathcal{X}$
$\mathcal{C}$	configuration space
$\mathcal{C}_{feas}$	feasible configuration space
$\mathcal{C}_{free}$	free configuration space
$\mathcal{C}_{obst}$	obstacle configuration space
$\mathcal{O}$	set notation for obstacles
$\mathcal{P}$	set of all robots trajectories
$\mathcal{R}$	set notation for robots
$\mathcal{U}$	allowable control input space
$\mathcal{W}$	set notation for world space where the robot moves
$\mathcal{X}$	state space
ω	angular velocity of the vehicle
π	trajectory
θ_r	vehicle orientation with respect to the x-axis

a_R, a_L	right and left wheel accelerations of the vehicle
C	number of collocation points
d_{safe}	safe margin between robot and obstacles
h	time interval between consecutive collocation points
h_r	vehicle's height
K	number of peripherals
L	distance between both actuated wheels of the vehicle
N	number of vehicles
O	number of obstacles
q	configuration of the vehicle/robot
R	robot
T	duration of trajectory
t	time when each state is reached
v	average linear velocity of the vehicle
v_R, v_L	right and left wheel velocities of the vehicle
w_r	vehicle's width
x_r, y_r	cartesian coordinates of the vehicle

Subscripts

$goal$	goal
$init$	initial
k	index of collocation point, $k \in \{0, ..., C-1\}$
max	maximum
min	minimum

Superscripts

i	robot index, $i = \{0, ..., N\}$
m	obstacle index

Glossary

2-D	Two Dimensional. 25, 26
AGV	Automated Guided Vehicle. 1, 2
AMR	Autonomous Mobile Robots. 2–6, 21, 25, 60, 63, 65–67, 73, 75, 76
APF	Artificial Potential Field. 14, 17
BLP	Binary Linear Programming. 18
BPM	Binary Public Map. 63
DDM	Dynamic Distributed Map. 22, 23
DMPC	Distributed Model Predictive Control. 21
DPM	Dynamic Public Map. 22, 23
EDT	Euclidean Distance Transform. 37, 38
FB	Flatness-Based. 18
FM2	Fast Marching Square. 15–17
FMM	Fast Marching Method. 15, 17
IPOPT	Interior Point Optimizer. xviii, 40–42, 48, 52
IST	Instituto Superior Técnico. 1, 65
LDM	Local Dynamic Map. 22, 23
LM	Local Map. 63
MILP	Mixed Integer Linear Program. 18
MIQP	Mixed Integer Quadratic Programming. 21

PM Public Map. 63

PRM Probabilistic Roadmaps. 12, 17

ROS Robot Operating System. 66

RRT Rapidly-exploring Random Tree. 13, 17, 30–33, 35, 39–42, 45, 52, 75, 76

RRT* Rapidly-exploring Random Tree Star. 13, 17

SDF Signed Distance Field. 37, 38, 43, 44, 52, 60, 61

SLSQP Sequential Least Squares Programming. xviii, 40, 41

SM Static Map. 62, 63, 65

SQP Sequential Convex Programming. 21

TEB Timed Elastic Band. 66, 76

TO Trajectory Optimization. 41, 42, 45–47, 49, 50, 52, 57

TP Trajectory Plannign. 41, 42, 45–50, 52

UAV Unmanned Aerial Vehicles. 2, 3

UPM Updated Public Map. 63

VO Velocity Obstacle. 21

Chapter 1

Introduction

1.1 Industrial Context

The massive technological development of the last century and the need to increase the scale of production have spawned a desire to automate the production process more and more. The advancements made in engineering fields have contributed to the increasing need to produce things more affordably and in less time, as well as to improve worker comfort by addressing ergonomics and workplace safety. Although this automation is nowadays applied in direct production tasks, it has not yet reached its full potential in the logistic flows that supply these production lines.

Logistic flows consider all processes, or sequences of processes, necessary to attain a goal in the shortest time possible, at the lowest cost and with the highest quality [1]. Intralogistics, which refers to the management, optimization, and distribution of products within the same warehouse or manufacturing unit, can be included in this context. Intralogistics focuses on reducing the manipulation and transport of materials in the factory's logistics flow, in order to make its process faster, more agile, cost-effective and have a significant impact on production outcomes [2].

The AGiLE project is a partnership between Instituto Superior Técnico (IST), Imeguisa and Volkswagen Autoeuropa, that focuses on developing a technological system to enable end-to-end automation of intralogistics flow in the Volkswagen Autoeuropa car factory. Nowadays, materials are transported inside the factory using Automated Guided Vehicles (AGVs) that follow a magnetic stripe on the factory floor. However, the AGVs solutions lack intelligence and communication capacities with the intralogistics flow in which they are included. In addition, AGVs use a "push" strategy when transporting materials in logistic facilities, i.e., the transport is always performed and the material is "pushed" to the assembly line regardless of its demand. This can lead to excessive and unnecessary transit, space occupation issues and the transport of material that are not immediately required. In addition, AGVs always execute a fixed trajectory and have no freedom of additional movement; they are unable to change their route to avoid obstacles or satisfy more than one path. Figure 1.1 illustrates a real situation in the Volkswagen

Autoeuropa facilities where AGVs operate.

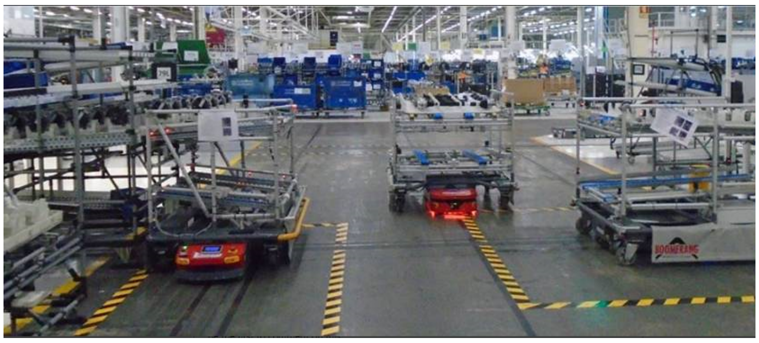
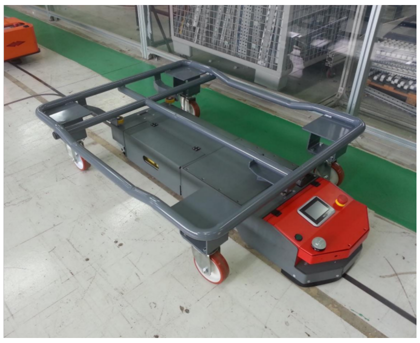



Figure 1.1: AGVs operating in the Volkswagen Autoeuropa facilities (left side). Example of an AGV transporting a rack and following a magnetic stripe of the floor (right side).

To overcome these limitations, the AGiLE project will integrate Autonomous Mobile Robots (AMRs) into the Volkswagen Autoeuropa manufacturing environment. These agents provide a more advanced solution, as they are able to communicate with one other, navigate freely and, consequently, adapt to dynamic scenarios. This multi-robot system will follow a non-centralized architecture, providing a scalable and resilient solution.

Without the need for human intervention, this solution will be able to manage and control an end-to-end intralogistics flow. This system should assign tasks to each AMR autonomously, considering factors such as availability, and collision avoidance. In conclusion, integrating these agents will enable the optimization of resources, taking into account the real-time requirements of an assembly line, always ensuring that it is accurately supplied and never interrupted.

1.2 Aerospace Motivation

Unmanned Aerial Vehicles (UAVs) have been one of the most challenging and promising technologies in the aerospace field in recent years. This progress has been directly affected by the exponential growth of both military and civilian UAVs applications in search and rescue, reconnaissance, area mapping, transportation, geological surveying, etc. [3–6]. The monitoring of any of these activities can be challenging, requiring UAVs to autonomously and cooperatively searching and tracking targets of interest [4, 5]. Consequently, effective decentralized strategies need to be implemented for UAVs to complete tasks cooperatively and time efficiently.

Regarding transportation, UAVs can be an alternative to existing logistics systems. They can operate above the production systems, thus extending the materials flow in a third dimension. Furthermore, a transportation system based on UAVs is highly flexible and allows fast transportation. Several other car manufactures have evaluated the viability of transporting components using drones within the factory [1].

Kendoul [3] identifies three functional technology areas as the core components of UAVs: guidance, navigation, and control. Among these functional areas, this dissertation mainly focuses on the guidance

[1] Audi. Logistics, Nov 2016. URL https://www.audi-mediacenter.com/de/audi-techday-smart-factory-7076/die-logistik-7082.

system, more specifically, it addresses the motion planning of a vehicle. The primary goal of an UAV optimized motion planning is to design a collision-free flight path to a target and, while also meeting the UAV performance requirements [6]. Therefore, the work developed in this dissertation can be extended for use in scenarios such as the generation of trajectories for a UAVs formation.

1.3 Project Outline

In this section, the system architecture that will be develop in the course of the AGiLE project is depicted. As illustrated in Figure 1.2, K peripherals scattered through different locations of the warehouse communicate with N robots that share information from a cloud.

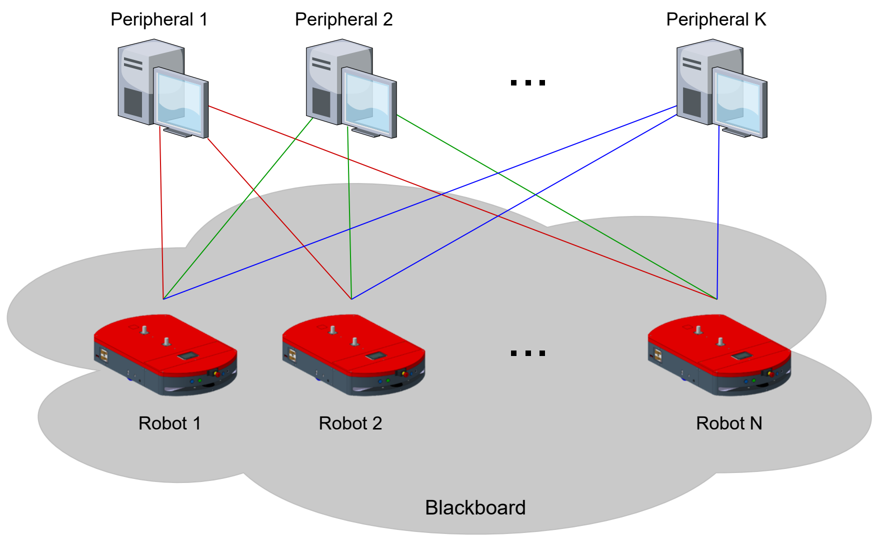


Figure 1.2: Interaction between K peripherals and N robots. These agents communicate and share information through a could, or Blackboard.

First, some terms must be defined:

- **Peripheral** - Agent responsible of recognizing when it is necessary to restock some component in the production line. When a task is to be executed, the peripheral should send a task request to all robots.
- **Robot** - Agent responsible for transporting components from an initial to a goal position. All N robots will receive a task request and they will decide in a decentralized manner which one will perform it.
- **Blackboard** - Each robot's system contains a copy of the updated blackboard, which stores information shared by all AMRs, such as:
  - updated map of the environment;
  - tasks requested by the peripherals to be auctioned;
  - tasks and respective trajectories that are already assigned to be executed.

- **Auction** - Deciding process. When a task is requested, each robot computes its bid and, then, shares it with the other robots. Each robot holds an auction with all the shared bids and decides which robot will performed the requested task.
- **Bid** - Each robot calculates one bid for each task. The bid considers the robot's autonomy and the time that it will take to start executing the task in auction.

The architecture of the system will be decentralized, i.e., the decision-making authority will not be concentrated on a single machine or agent. Thus, each robot makes its own decisions based on local information. This architecture decomposes the multi-robot system problem into a set of single-agent ones. For instance, each robot plans its own route, while aiming at a globally optimal and safe solution. Additionally, this approach is robust, since the system will not fail even if communication with one robot fails. The decentralized method is considered to be an adequate approach for this dissertation problem because it is scalable, robust, and reduces the system's complexity.

1.3.1 Sequential Description

The sequence diagram presented in Figure 1.3 depicts the interactions between a peripheral and a single robot over time, to assign a specific task. The robot system is composed of five elements: four function blocks, namely the Task Manager, Map Manager, Task Executer and Trajectory Optimizer; and the fifth element, the Blackboard, which is a copy of the data shared by all robots, as mentioned previously.

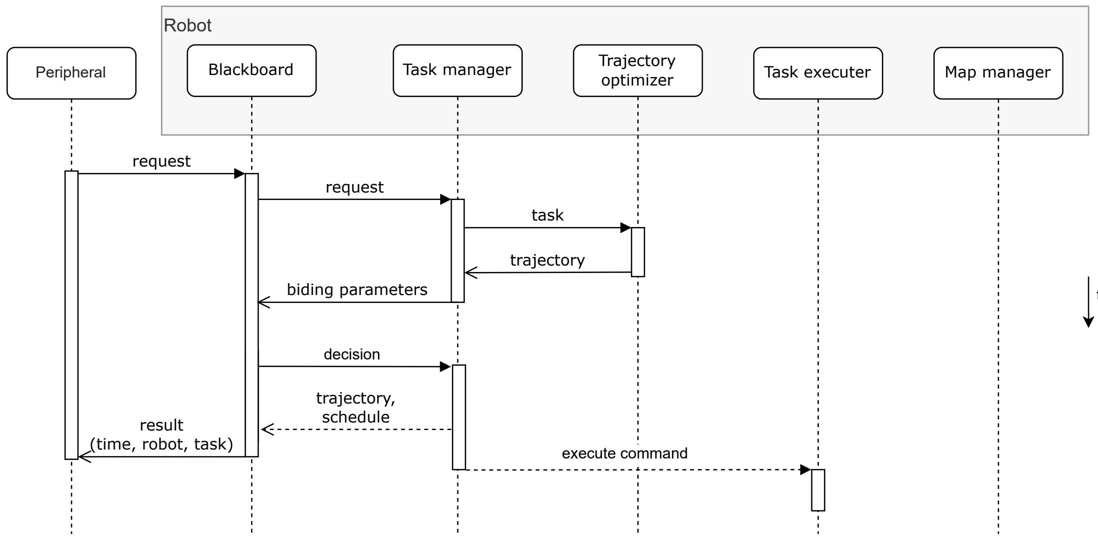


Figure 1.3: Sequence Diagram for task assignment. A peripheral requests a task and an auction process is held to decide on the winning robot that will execute the assigned task.

The four function blocks presented above can be defined as:

- **Task manager** - Performs an auction to decide which AMR will execute the requested task.
- **Map Manager** - Updates a map whenever there is a significant change in the environment and checks if the AMR trajectory needs to be recalculated.

- **Task Executer** - Executes the guidance of the AMR.

- **Trajectory Optimizer** - Creates an optimized trajectory to complete the task.

Firstly, the peripheral requests a new task. The request is sent to the Blackboard, which stores the task request before forwarding it to the Task Manager. This element will solicit an optimized trajectory from the Trajectory Optimizer. With this trajectory, the Task Manager will hold an auction that, taking into consideration the biding parameters of all robots, decides which one will perform the specific task. When a decision is made, the Task Manager of the winning robot sends the chosen trajectory and its respective schedule to the Blackboard. Then, when the task needs to be performed, a command is sent to the Task Executer, so that the AMR can begin following the trajectory.

Nevertheless, facility logistics environments are not static, due to factory operations that can create new obstacles. A static reference map that contains static features of the environment, such as corridors and fixed machinery, will be available beforehand. This reference map of the factory should be updated with changes that arise in the environment. Therefore, it is possible to improve the robots' path planning and the coordination of the multi-robot system. Figure 1.4 presents the sequence diagram for dealing with a map update in a decentralized manner.

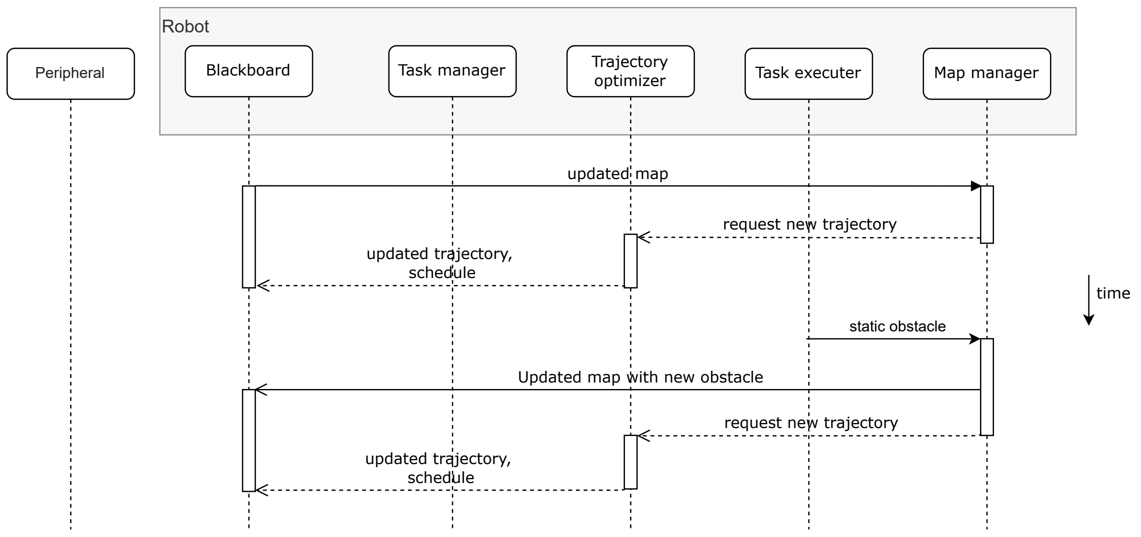


Figure 1.4: Sequence Diagram for map updates. Two distinct scenarios are considered: another robot updates the shared map; or the considered robot detects a significant change in the environment.

The above diagram depicts two distinct situations. In the first, a map that has been updated by another robot is uploaded to the AMR's Blackboard. In the second scenario, the robot can detect a significant change in the environment while performing a task. Then, the Map Manager will generate an updated map to be saved in the Blackboard and shared with the other AMRs. In both situations, the Map Manager should decide whether or not rerouting is required. In the affirmative case, the Trajectory Optimizer receives a request for planning a new trajectory.

1.3.2 Functional Architecture

In Figure 1.5 the functional architecture implemented in each AMR is presented. As previously described, this system can be divided into one data storage (Blackboard) and four function blocks (Task Manager, Map Manager, Task Executer and Trajectory Optimizer).

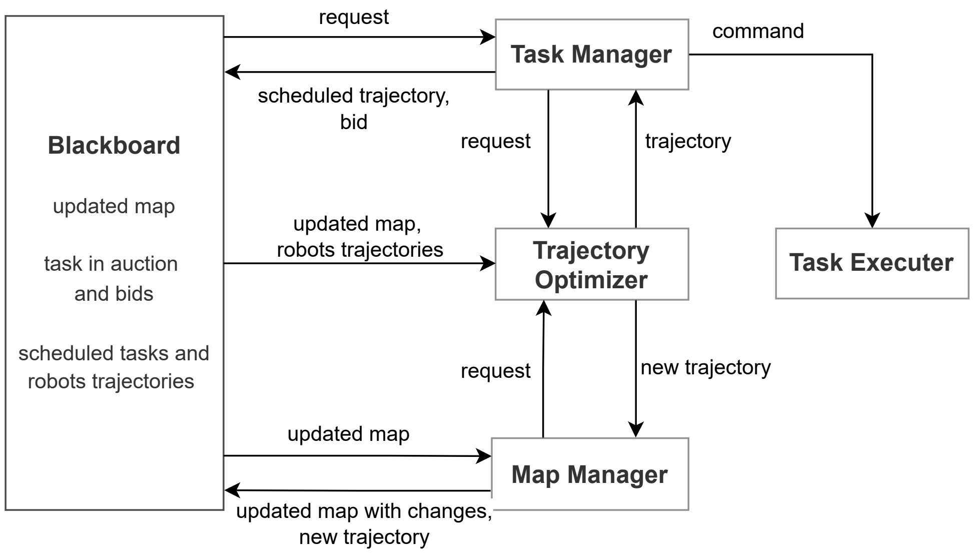


Figure 1.5: Functional Architecture of a single AMR. The data storage (Blackboard) and four function blocks (Task Manager, Map Manager, Task Executer and Trajectory Optimizer) are represented, as well as the inputs and outputs of each module.

The Task Manager takes as an input a request for a new task. To compute the bid for the auctioned task, this function block uses the trajectory calculated in the Trajectory Optimizer. The Trajectory Optimizer takes as input a requested task and provides as an output an optimized trajectory to complete it. The bid of each AMR is shared between all robots and the Task Manager determines which robot wins the auction. A command is sent to the Task Executer of the winning AMR to initialize the guidance and carry out the task. While this AMR is following the defined trajectory, two different scenarios can arise: the AMR detects a new static obstacle; or another robot discovers a new obstacle. In the first scenario, as explained before, the Map Manager will take as inputs the static obstacle coordinates detected by the respective AMR and creates an updated map for all robots to share. For the second scenario, the Map Manager will receive an updated map from the other robots. In both situations, the local map of the AMR is updated and it is determined whether, given the newly updated map, any trajectories of the AMR must be redefined.

This dissertation focuses primarily on the decentralized trajectory optimization problem. Therefore, all AMRs integrated in the industrial fleet should communicate with each other, in order to contribute to updating the factory's static map and generating optimized trajectories. Consequently, the Trajectory Optimizer and Map Manager are the function blocks that will be further analysed and developed in this dissertation. On the other hand, the Task Manager and Task Executed will be implemented by J. Tavares [7].

1.4 Problem Statement

The problem of interest is the general optimization of a multi-vehicle system, with a particular emphasis on trajectory planning. Each agent in a fleet of N vehicles is assumed to be autonomous and capable of either moving or standing still. The velocity and position of adjacent agents as well as the locations of several obstacles can be determined using equipped sensors. Additionally, each agent is aware of the known static map of the environment, the trajectories being executed by the other agents and the tasks and trajectories already assigned to all of them. The tasks will be requested asynchronously. Therefore, these trajectories will be generated in a decentralized and sequential manner.

To perform a requested task, the Trajectory Optimizer function block of each agent should take an initial and final pose and plan a trajectory to complete it. The planning must consider the agent's availability, as it may already be scheduled to perform other tasks. To generate a collision-free trajectory, each agent must be aware of the locations of all other agents at every instant. Each trajectory should be an optimized solution that minimizes a cost function that depends on parameters to define in the course of this work.

An autonomous robot must perceive and act accordingly with its environment, as this is the foundation for optimizing its trajectory. Since the environment under study is subject to change, it is necessary to distinguish between static and dynamic obstacles. Static obstacles are fixed objects from the agent's perspective. On the other hand, the location of dynamic obstacles changes in the agent's field of view. In the Map Manager function block this distinction between the two types of obstacles must be made and the static obstacles must be taken into account when representing the current environment. Let us define the following terminology:

- **Static Map** - Map built initially. It contains static features of the environment like walls or fixed objects.

- **Temporary Map** - Map built by an agent upon detecting significant changes in the environment.

- **Updated Map** - Map updated by merging temporary maps with the static map.

Initially, all vehicles will own a static map. When an agent detects a significant change, i.e., appearance or disappearance of a static obstacle, the Map Manager function block must generate a temporary map. This temporary map should be sent to the Blackboard and distributed to the other agents. Then, an updated map must be created by merging all maps, so that robots can plan their trajectories using all of the shared information. By considering static obstacles and updating the known initial map of the environment, the agent's trajectory planning and the coordination of the multi-robot system can be improved.

1.5 Objectives

The main goal of this dissertation is to develop an optimized trajectory planning algorithm, that is capable of generating feasible and safe trajectories for a fleet of industrial mobile robots performing factory logistics tasks. In order to accomplish this, the following objectives must be met:

- Study the concepts and assess past research regarding motion planning, multi-robot trajectory planning, trajectory optimization and dynamic map update.
- Define a motion planning algorithm that generates optimized trajectories.
- Outline a map updating technique to be implemented.
- Integrate the developed solution on the software architecture of a real robot fleet.
- Finally, throughout the stage of implementation, test and validate iteratively the developed solution.

1.6 Contributions

This dissertation presents the following contributions:

- Implementation of a kinodynamic trajectory planning algorithm.
- Novel formulation for multi-robot trajectory optimization with only one constraint regarding obstacle avoidance, which includes both static and dynamic obstacles.
- Simulation and evaluation of the algorithm performance.
- Novel reformulation of the objective function to obtain trajectories that imposes vehicles to drive on the right side of the corridors.
- An authored work entitled of “Sequential Multi-Agent Trajectory Optimization Using Signed Distance Fields”, which is presented in the extended abstract, was recently submitted for presentation on the 2023 IEEE International Conference on Robotics and Automation (ICRA) to be held in London, England, on 29th May to 2nd June 2023. An accompanying video was also submitted illustrating the results obtained [2].

1.7 Dissertation Outline

The dissertation is structured as follows:

- **Chapter 2 -** Reviews the relevant literature for the work to be developed in the dissertation. An overview of path planning algorithms is given. The trajectory optimization problem is examined. Methods proposed for multi-robot trajectory planning are presented. Finally, map merging techniques are analysed, followed by examples of map sharing techniques.

[2]youtube video URL: https://youtu.be/uno3QLNpO9Y

- **Chapter 3 -** Introduces the proposed algorithm to solve a single-robot trajectory optimization problem. Additionally, the model of the vehicle under study is defined.
- **Chapter 4 -** Presents the trajectory planning method proposed to search for feasible and collision-free routes.
- **Chapter 5 -** Formulates the single-robot optimization problem.
- **Chapter 6 -** Formulates the multi-robot optimization problem and evaluates the algorithm's performance.
- **Chapter 7 -** Presents adjustments done to the proposed algorithm to satisfy the real-industrial environment.
- **Chapter 8 -** The proposed trajectory planning algorithm is tested in a real-industrial environment.
- **Chapter 9 -** Gives a summary of the work developed, main conclusions and outlines the future work.

Chapter 2

Background

In this chapter, a literature review is conducted for the work that is developed in the dissertation. In Section 2.1, the fundamental concepts of motion planning and an overview of path planning algorithms are presented. Next, in Section 2.2 the trajectory optimization problem is defined and several approaches to solve it are analysed. In Section 2.3 methods for multi-robot trajectory planning are proposed. Finally, in Section 2.4 a preliminary analysis on map merging methods is presented, followed by examples of map sharing techniques.

2.1 Motion Planning

Motion planning is a topic widely researched and many algorithms have been developed to fulfil the goal of finding the optimal motion plan. Firstly, path planning must be distinguished from trajectory planning [8]. Informally, a path is a spatial construct with no notion of time that describes how to go from an initial to a goal point. In contrast, a trajectory provides a notion of time by assigning a time constraint to a geometric path. Trajectory planning assumes that the output is continuous in time and must be able to ensure control limitation of the vehicle, thereby affecting robot kinematics and dynamics.

This section begins by introducing the configuration space used to formulate the motion planning problem. Afterwards, an overview of the major contributions to solving it is provided.

2.1.1 Configuration Space

As it is presented in [9], the motion planning problem can be mathematically formalized. Let define the space of all configurations as the configuration space, $\mathcal{C}$. Consider an agent, R, moving in an environment $\mathcal{W} \in R^2$, populated by a set of M obstacles $\mathcal{O} = \{O^1, ..., O^M\}$.

The configuration of R in $\mathcal{W}$ that specifies completely its position and orientation in the system is defined as $q \in C$. The obstacle configuration space $\mathcal{C}_{obst} \subset \mathcal{C}$ is defined as the set of configurations q at which the robot intersects any obstacle in the set $\mathcal{O}$. Contrarily, the obstacle-free configuration space is denoted by $\mathcal{C}_{free} = \mathcal{C} \backslash \mathcal{C}_{obst}$. Finally, a path planning problem can be defined as finding a continuous and collision-free path $\pi : [0, 1] \rightarrow C_{free}$ between the initial, $q_{init} = \pi(0)$, and goal configurations, $q_{goal} = \pi(1)$.

2.1.2 Motion Planning Methods

In this section, advantages and disadvantages of several motion planning methods are discussed. The methods addressed are the following: Decomposition Graph-based Methods, Sampling-based Methods, Artificial Potential Field and Fast Marching Method. Figure 2.1 depicts the motion planning method families addressed.

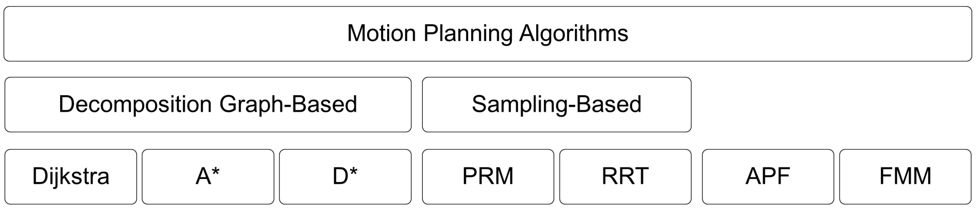
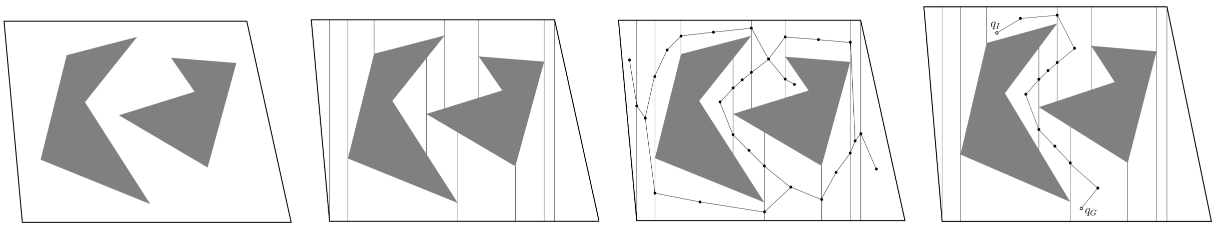


Figure 2.1: Motion planning algorithms.

Decomposition Graph-based Methods

Decomposition graph-based methods [10] divide or decompose $\mathcal{C}_{free}$ into a finite number of regions, cells. Then, it is defined a connectivity graph $G = (N, E)$, which represents the adjacency relations between cells. That is, the robot configurations from the free space are represented by the nodes, N, of the graph, and there is an edge or arc, E, joining two nodes if and only if the corresponding cells of the nodes are adjacent. In Figure 2.2 is illustrated as example of a decomposition graph-based method, in which vertical cell decomposition is used. The path planning problem can be solved using graph planning techniques. Some of the most used examples of these techniques are: Dijkstra, A* and D*



(a) Polygonal obstacle region. (b) Vetical cell decomposition. (c) Derived roadmap with nodes connected by edges. (d) An example solution path.

Figure 2.2: Example of decomposition graph-based method that uses vertical cell decomposition [9].

Developed by Edsger W. Dijkstra [11], the **Dijkstra** algorithm tries to find the shortest path in a graph. This algorithm is classified as complete, i.e., finds a solution if one exists. Nonetheless, the computational cost grows quadratically as the problem complexity increases. Hart's **A*** algorithm [12] is also complete. Contrary to Dijkstra, this algorithm employs a heuristic estimation of the cost to the goal node and, consequently, does not go through the entire graph when searching for a solution. Hence, **A*** converges faster in larger environments. Introduced by Stentz [13], **D*** algorithm is a dynamic variant of **A***, since the weights of the edges can change as the robot goes through the path and new information is found. Nevertheless, decomposition graph-based methods can only guarantee optimality in discretized spaces.

Sampling-based Methods

Sampling-based algorithms are a successful approach to robotic motion planning problems. These methods sample random configurations to map the environment, i.e., create a set of paths searched in $\mathcal{C}_{free}$. The sampling-based algorithms can be classified between multiple-query and simple-query approaches. For multiple-query algorithms a graph is created, or roadmap, that can be used to obtain efficient solutions for many pairs (q_{init}, q_{goal}). Therefore, once the roadmap has been generated with all the possible paths in the environment, it can be reused for all agents in the system. On the other hand, for single-query algorithms, a single pair (q_{init}, q_{goal}) is given and a solution is searched until one is found, or returns failure. Next, the Probabilistic Roadmaps and the Rapidly Exploring Random Trees which are, respectively, included in the multiple-query and single-query families, are presented.

The **Probabilistic Roadmaps** (PRM) algorithm [14], starts by generating a roadmap, from an initial empty graph $G = (N, E)$. The pseudocode for generating a roadmap is outlined in Algorithm 1. A new random free configuration q_{rand} is sampled and added to the set of configurations N (lines 4 and 5). As illustrated in Figure 2.3, the nearest nodes to q_{rand} are chosen (line 6). In case of a collision-free edge between a near configuration and the random configuration, an edge is added to E (line 9). This process is repeated until a maximum number of configurations are generated or until a time limit is reached. In order to determine a path between the starting and goal configurations, a graph search algorithm is applied, such as Dijkstra or A*.

Algorithm 1 PRM - roadmap construction

```
    Input: configuration space $\mathcal{C}$
    Output: PRM graph $G = (N, E)$
 1: $N \leftarrow \emptyset$
 2: $E \leftarrow \emptyset$
 3: for $iter < max\_iter$ do
 4:     $q_{rand} \leftarrow$ RandomNode($\mathcal{C}$)
 5:     $N \leftarrow N \cup q_{rand}$
 6:     $N_{near} \leftarrow$ a set of candidate neighbours of $q_{rand}$
        chosen from $N$
 7:     for all $q_{near} \in N_{near}$ do
 8:         if CollisionFree($q_{near}, q_{rand}$) then
 9:             $E \leftarrow E \cup \{(q_{near}, q_{rand})\}$)
10:         end if
11:     end for
12: end for
```

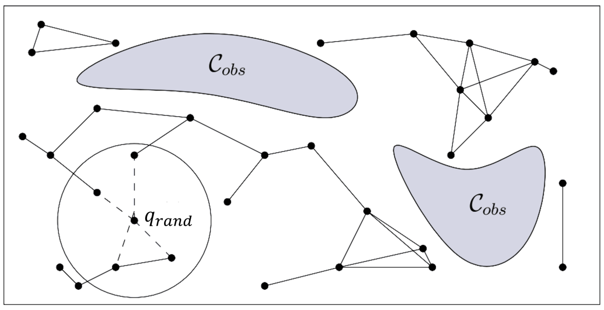


Figure 2.3: Roadmap built incrementally, by attempting to connect each new sample, q_{rand}, to nearby nodes in the roadmap [9].

One of the advantages of PRM is the fact that it allows to explore large environments with a low computational cost. However, this method is probabilistically complete, i.e., the completeness of the method depends on the number of samples, meaning that the probability of finding a solution converges to 1 when is given an infinite amount of time.

Introduced by LaValle in [9], **Rapidly-exploring Random Tree** (RRT) is an efficient single-query algorithm. It constructs a single tree $\mathcal{T} = (N, E)$ starting from an initial configuration, q_{init}. The basic RRT construction is given in Algorithm 2. A random sample, q_{rand} from $\mathcal{C}$ is taken (line 3) and an attempt is made to connect it to the nearest state of the tree, q_{near} (line 4). The function $NewConfig$ makes a motion towards q_{rand} with some fixed incremental distance ϵ and creates q_{new}. If it is ensure a collision-free connection between q_{new} and q_{near}, q_{new} is added to the tree (lines 6 to 8). This process is repeated until a maximum number of iterations is reached. Figure 2.4 depicts the RRT expanding operation described.

Algorithm 2 Simple RRT

```
    Input: initial configuration $q_{init}$, configuration space $\mathcal{C}$
    Output: RRT tree $\mathcal{T} = (N, E)$
 1: $N \leftarrow \{q_{init}\}$
 2: for $iter < max\_iter$ do
 3:     $q_{rand} \leftarrow$ RandomNode($\mathcal{C}$)
 4:     $q_{near} \leftarrow$ NearestNeighbor($q_{rand}, \mathcal{T}$)
 5:     $q_{new} \leftarrow$ NewConfig($q_{near}, q_{rand}$)
 6:     if CollisionFree($q_{near}, q_{rand}$) then
 7:         $N \leftarrow N \cup \{q_{new}\}$
 8:         $E \leftarrow E \cup \{(q_{near}, q_{new})\}$
 9:     end if
10: end for
11: return $\mathcal{T}$
```

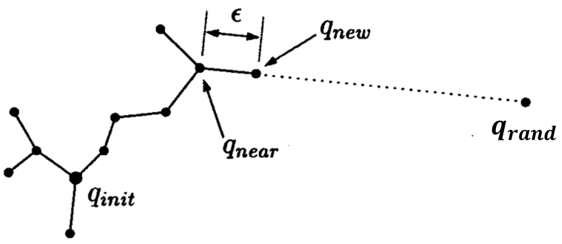


Figure 2.4: RRT expanding operation [15].

Nevertheless, this algorithm is also probabilistically complete. To tackle this limitation, a variant of the RRT algorithm, the Bidirectional RRT, was developed. This algorithm grows two trees, one rooted at the initial state and another at the goal state, in order to connect them and increase the probability of finding a solution.

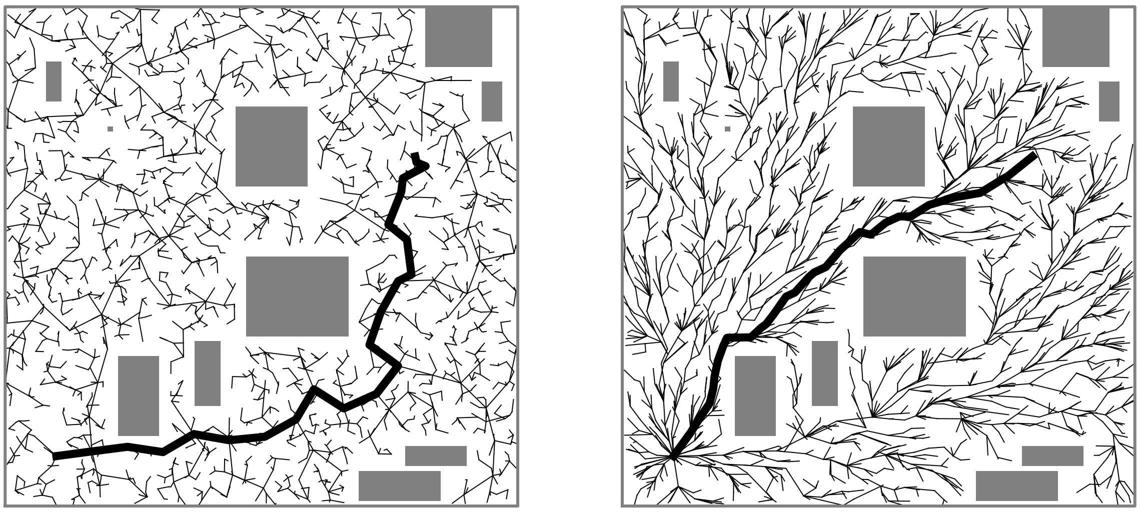
Another drawback of the RRT is that this solution is not optimal, meaning that the generated path does not minimize the distance between the start and goal positions, as illustrated in Figure 2.5a. To overcome this challenge, Karaman and Frazzoli popularized an extension named **Rapidly-exploring Random Tree Star** (RRT*) [16]. The pseudocode for the RRT* is presented in Algorithm 3. Similarly to the RRT, in each iteration, the robot configurations are sampled from the entire configuration space. A random node is connected to the nearest node of the tree, as described previously. Then, it is made an attempt to find a new parent node of q_{new}, q_{min}, that holds a minimum cost (lines 8 and 9). In the affirmative case, the new parent node is then connected to q_{new} (line 11). Additionally, it is performed a rewiring process, in which it is verified if q_{new} holds a smaller cost as a parent rather than as a child for any neighbouring node (line 13). If this is true, the tree can be rebuilt for a minimum cost. Finally, the algorithm continues with a new iteration.

In fact, the RRT* is more difficult to construct, compared to the simple RRT. Nevertheless, the RRT* algorithm provides an asymptotic optimal path solution, i.e., it converges to the optimal solution with an infinite number of iterations, as depicted in Figure 2.5b.

Algorithm 3 RRT*

```
Input: initial configuration $q_{init}$, configuration space $\mathcal{C}$
Output: RRT* tree $\mathcal{T} = (N, E)$

1: $N \leftarrow \{q_{init}\}$
2: for $iter < max\_iter$ do
3:     $q_{rand} \leftarrow$ RandomNode($\mathcal{C}$)
4:     $q_{near} \leftarrow$ NearestNeighbor($q_{rand}, \mathcal{T}$)
5:     $q_{new} \leftarrow$ NewConfig($q_{near}, q_{rand}$)
6:     if CollisionFree($q_{near}, q_{rand}$) then
7:         $N \leftarrow N \cup \{q_{new}\}$
8:         $Q_{near} \leftarrow$ Near($q_{new}$)
9:         $q_{min} \leftarrow$ argmin$\{q \in Q_{near}$: cost($q$)+ c($q, q_{new}$)$\}$
10:        if CollisionFree($q_{min}, q_{new}$) then
11:            $E \leftarrow E \cup \{(q_{min}, q_{new})\}$
12:        end if
13:        $q_{child} \leftarrow$ argmin$\{q \in Q_{near}$: cost($q_{new}$)+ c($q_{new}, q$)$\}$
14:        if CollisionFree($q_{new}, q_{child}$) then
15:            $E \leftarrow (E \setminus \{(\text{parent}(q_{child}), q_{child})\}) \cup \{(q_{new}, q_{child})\}$
16:        end if
17:    end if
18: end for
19: return $\mathcal{T}$
```



(a) Rapidly-exploring Random Tree (b) Rapidly-exploring Random Tree Star

Figure 2.5: A Comparison between the (a) RRT and (b) RRT*. Both algorithms were run with a maximum iteration of 3000 samples.

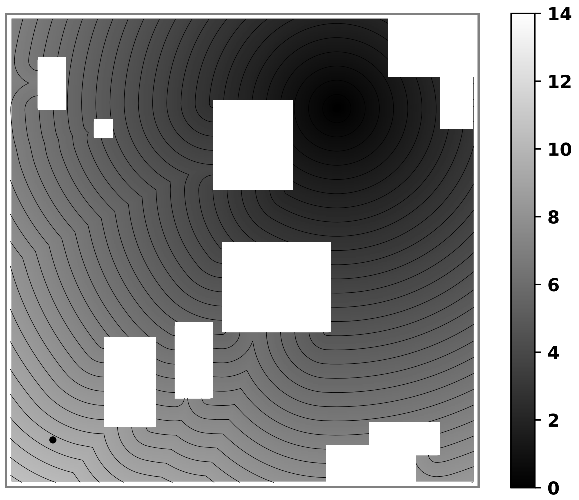
Artificial Potential Field

Introduced by Khatib [17], the **Artificial Potential Field** (APF) approach is one of the most popular techniques for path planning. The goal configuration is typically modeled as an attractive potential, whereas obstacles generate repulsive forces to prevent agents from colliding with them. Consequently, the sum of these forces, the total potential, causes the agent to move along the optimal local trajectory.

One major drawback of the APF approach is that this algorithm is incomplete: a local minima can compromise the attainment of the goal. However, this problem can be overcome by using navigation functions, such as harmonic potential functions [18, 19].

Fast Marching Method

Wavefront propagation methods, such as the **Fast Marching Method** (FMM), can be an alternative approach to the path planning problem. The FMM is a particular case of Level Set Methods, created by Osher and Sethian [20]. Intuitively, FMM exploits a wavefront technique derived from a source point. In path planning, a binary gridmap is considered, in which the $\mathcal{C}_{obst}$ is valued as zero, while $\mathcal{C}_{free}$ is valued as one. If a wave is expanded from the goal configuration q_{goal} to the initial one q_{init}, then at the obstacles, the wave expansion speed is zero, as it can not go through the obstacles. On the other hand, in free space, the wave expansion speed is constant and equal to one. Hence, the wave originates a field that has only one global minima at the source. By following the minimum gradient direction from the initial point to the source, goal point, a solution can be obtained. This algorithm is complete, since a solution is always reached whenever there is a valid path. As mentioned in [21], FMM achieves the shortest path in length. Nevertheless, as depicted in Figures 2.6a-b, this solution might not be safe because of its proximity to obstacles.



(a) Fast marching method wave expansion.

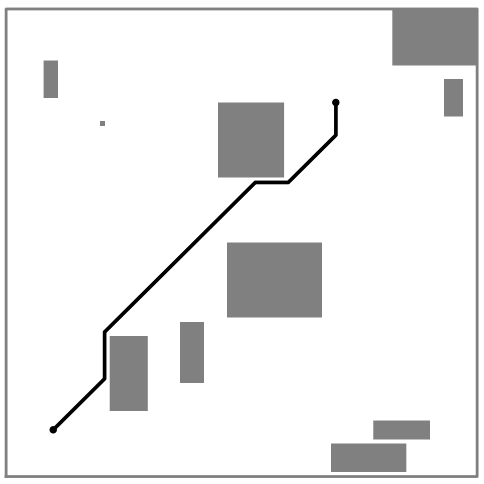
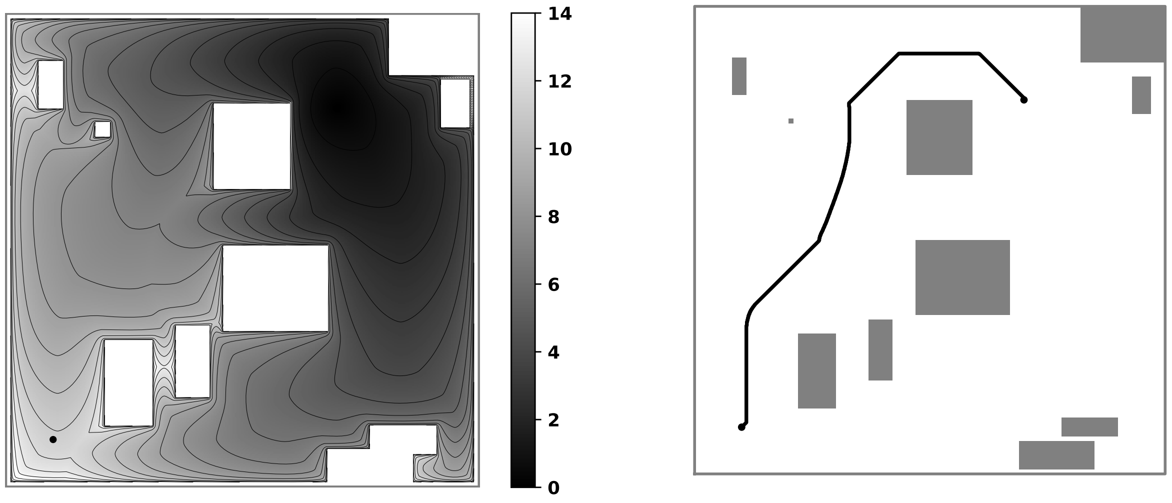

(b) Path obtained with fast marching method.

Figure 2.6: Example of path generated using FMM. The wave expansion sourced at the goal location has a constant speed equal to one (black is the closest to the source and light grey is the furthest).

To tackle this, a new variant was introduced: **Fast Marching Square** (FM2) [22]. The pseudocode to generate a path using FM2 is presented in Algorithm 4. This method starts by taking the gridmap, labelled as mentioned above, and applying to it the FMM considering the obstacles as sources. The result is a slowness map, which represents the potential field of the environment. This map can also be seen as the maximum allowed speed of the robot at each cell. Thus, the path can be determined using once again the FMM, but instead of tacking a constant expansion speed as before, the speed is given by the slowness map. The FM2 assumes that the robot moves at the maximum allowed speed at every point. Nevertheless, this algorithm is not optimal since it tries to keep the computed path as far away from the obstacles as possible, as illustrated in Figures 2.7a-b.

Algorithm 4 FM2

```
    Input: gridmap $G$, goal configuration $q_{goal}$, obstacle grid point $obstacles$
    Output: gridmap $G$ with time value set for all cells
1: $slowness\_map \leftarrow$ FMM($G, obstacles$)
2: $fm2\_map \leftarrow$ FMM($slowness\_map, q_{goal}$)
3: $path \leftarrow$ MinimumGradientDirection($fm2\_map, q_{initial}, q_{goal}$)
4: return $path$
```



(a) Fast marching square wave expansion.

(b) Path obtained with fast marching square.

Figure 2.7: Example of path generated using FM2. The wave expansion sourced at the goal location has variable speed given by the slowness map (black is the closest to the source and light grey is the furthest).

Table 2.1 depicts the comparison of the methods analysed in this section. The characteristics considered are:

- **Completenes -** It is guaranteed that the algorithm finds a solution if exists at least one.
- **Optimality -** The solution found optimizes performance in some specified manner.
- **Computational cost -** The number of resources required to run the algorithm, such as time and memory resources.
- **Searching Approach -** A deterministic algorithm, given a specific input, always produces the same output. A heuristic algorithm can solve a problem faster and more efficiently by sacrificing optimality, accuracy, precision, or completeness for speed. A randomized algorithm, given a specific input, does not produce the same output.

Table 2.1: Path planning algorithms characteristics.

Algorithm	Completeness	Optimality	Computational Cost	Approach
Decomposition graph-based	complete	optimal[1]	high	deterministic or heuristic
PRM	complete[2]	non-optimal	low	randomized
RRT	complete[2]	non-optimal	low	randomized
RRT*	complete[2]	asymptotic optimal	low	randomized
APF	incomplete	optimal	low	deterministic
FMM	complete	non-optimal	high	deterministic
FM2	complete	non-optimal	high	deterministic

[1] For discretized space.

[2] For infinite iterations.

2.2 Trajectory Optimization

In the previous section, the time dimension in motion planning problems was not considered. Nevertheless, it is not only important to consider global constraints, which are imposed by obstacles. In fact, robots often have local constraints, which can be considered as limits in velocities or accelerations at every point due to kinematic and dynamic restrictions. Thus, the present section addresses trajectory optimization, which is an established method in robotics that generates a high-quality motion from an initial trajectory that may be in-collision or dynamically infeasible. These techniques represent mathematically a robot with motion equations. Trajectory optimization determines the dynamical system that optimize a specified performance while satisfying a set of constraints. According to [23] the optimization problem can be laid out as:

$$\begin{aligned}
\text{Minimize} \quad & J(t_0, t_f, \mathbf{x}(t_0), \mathbf{x}(t_f)) + \int_{t_0}^{t_f} w(t, \mathbf{x}(t), \mathbf{u}(t))\, dt & & \quad (2.1)\\
\text{w.r.t.} \quad & t_0, t_f, \mathbf{x}(t), \mathbf{u}(t)\,, & & \\
\text{subject to} \quad & \dot{\mathbf{x}}(t) = f(t, \mathbf{x}(t), \mathbf{u}(t)) & & \text{system dynamics}\\
& h(t, \mathbf{x}(t), \mathbf{u}(t)) \leq 0 & & \text{path constraint}\\
& g(t_0, t_f, \mathbf{x}(t_0), \mathbf{x}(t_f)) \leq 0 & & \text{boundary constraints}\\
& \mathbf{x}_{min} \leq \mathbf{x}(t) \leq \mathbf{x}_{max} & & \text{path bound on state}\\
& \mathbf{u}_{min} \leq \mathbf{u}(t) \leq \mathbf{u}_{max} & & \text{path bound on control}
\end{aligned}$$

In general the cost function of the optimization problem presented in Equation (2.1) includes two terms: a boundary objective $J(.)$ and a integral objective $w(.)$, taking as decisions variables the initial and final time (t_0, t_f) and the state and control trajectories $(x(t), u(t))$. The optimization is subject to a variety of limits and constraints. The first constraint presented bellow is the system dynamics which is usually non-linear and describes how the system changes in time. The path constraint, $h(.)$, is formulated to avoid obstacles. Additionally, the boundary constraint, $g(.)$, limits the robot to the initial and final states. Finally, the two last constraints represent the physical limits of the system applied throughout the entire trajectory, such as limits on torque and speed. The limits in the state and control trajectories can be expressed as (x_{min}, x_{max}) and (u_{min}, u_{max}), respectively.

The taxonomy that classifies the trajectory optimization techniques addressed in this section is illustrated in Figure 2.8. These techniques can be divided into Linear Programming and Optimal Control. In the following subsections, an overview of both approaches is made.

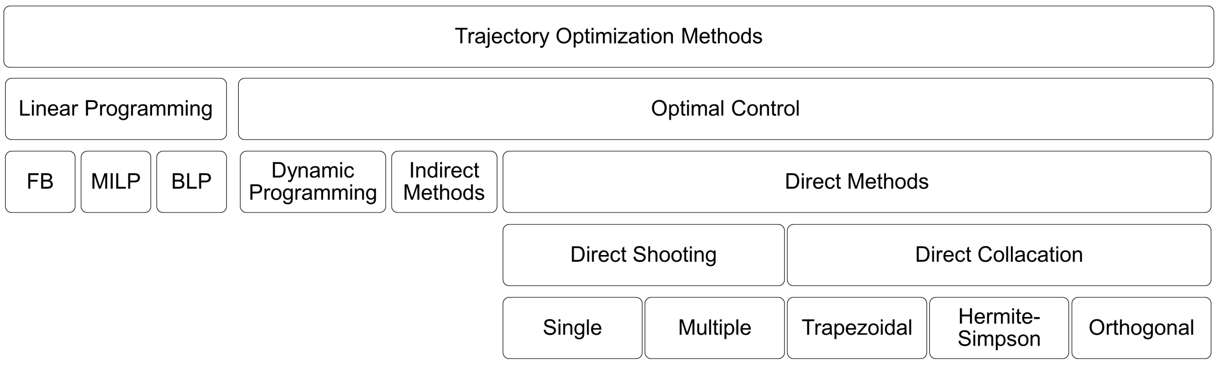


Figure 2.8: Trajectory optimization methods.

2.2.1 Linear Programming

Linear algorithms describe the environment, given kinematic and dynamic constraints. In linear programming, the cost function and its constraints are all linear equations. Proposed by Chamseddine et al. [24], the **Flatness-Based** (FB) linearizes the non-linear kinodynamic constraints of the system to form a simpler formulation. **Mixed Integer Linear Program** (MILP) [25] has a strong modelling capability to describe almost all the information. Although its high computation complexity is a significant drawback. **Binary Linear Programming** (BLP) [26] is a special case of linear programming where the variables are only 0 or 1.

2.2.2 Optimal Control

According to [27], the optimal control framework might be more suitable for formulating a problem with both global and local constraints. The solution in this framework is a trajectory encoded as a sequence of states and controls that optimizes a given objective function. As per [23], the techniques for solving optimal control problems can be divided into three categories: Dynamic Programming, Indirect and Direct Methods.

Dynamic programming solve Hamilton-Jacobi-Bellman equations over the entire state space. This is an effective method for solving optimal control problems in unconstrained low-dimensional systems, but not for high-dimensional ones.

An **indirect method** analytically constructs the necessary and sufficient conditions for optimality. These conditions are then discretized and numerically solved. Indirect methods can achieve more accurate solutions, at the expense of being more difficult to construct and solve.

On the other hand, a **direct method** discretizes the trajectory optimization problem and converts it into a non-linear programming problem. Thus, since direct optimization methods usually do not require an analytical analysis of the problem, they are the most used methods for trajectory optimization problems. In this subsection, a brief description of direct trajectory optimization methods, such as direct shooting and direct collocation, is provided.

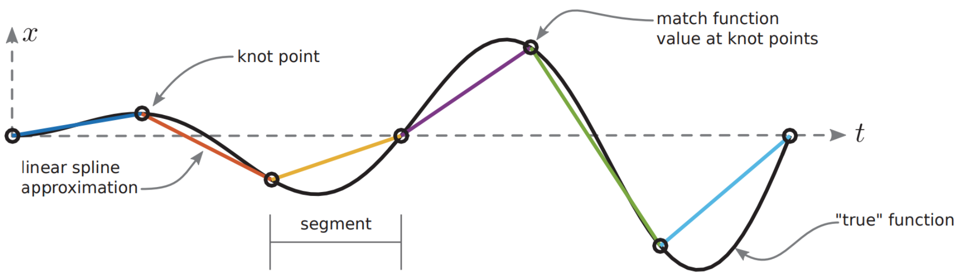
Direct Shooting methods

Single shooting is the simplest method to solve a trajectory optimization problem and can be used with either direct or indirect formulations. This method approximates the trajectory using a simulation. A set of parameters for the trajectory is chosen and then it is simulated and checked if the target is reached. The route is represented as a single segment with a single constraint, in which the final configuration of the trajectory must be the goal configuration. This approach is applicable when the control is simple and there are few path constraints, such as space flight [28]. A disadvantage of single shooting methods is that the relationship between decision variables, objective function and constraints is often highly non-linear and cannot be well approximated by the model [23].

Additionally, **multiple shooting** is an extension of single shooting that tends to be more robust and adequate to solve more challenging trajectory optimization problems. Instead of representing the entire trajectory as a single simulation, in multiple shooting the trajectory is divided into many shorter segments and employs single shooting for each segment. As the segments get shorter, the relation between decision variables, objective function and constraints become more linear.

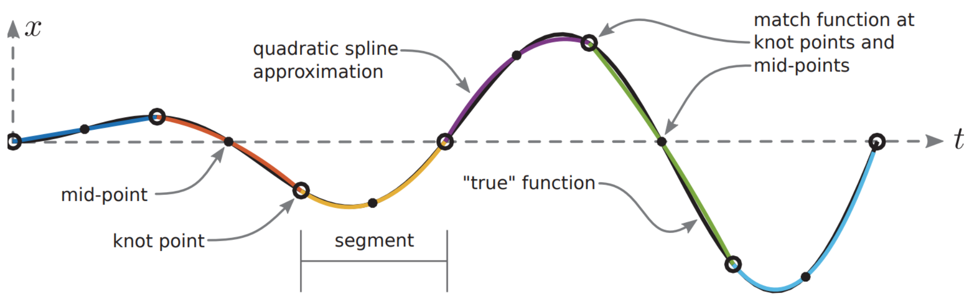
Direct Collocation methods

Direct collocation methods try to discretize a continuous trajectory optimization problem by approximating the state and control trajectories using polynomial splines (function that is defined piecewise by polynomials). Two commonly used collocation techniques are the **trapezoidal collocation** and **Hermite-Simpson collocation**. According to Betts [27], the trapezoidal collocation approximates the objective function and system dynamics as piecewise linear functions, using trapezoidal quadrature. On the other hand, Hermite-Simpson collocation approximates the objective function and system dynamics as piecewise quadratic functions. As an advantage, the state trajectory with the Hermite-Simpson collocation method is a cubic Hermite spline, which has a continuous first derivative [23]. Figure 2.9 shows a comparison of trapezoidal and Hermite-Simpson collocation methods.



x
knot point
match function value at knot points
t
linear spline approximation
segment
"true" function

(a) Trapezoidal collocation approximates the function with piecewise linear functions.



x
quadratic spline approximation
match function at knot points and mid-points
t
mid-point
knot point
segment
"true" function

(b) Hermite-Simpson collocation approximates the function with piecewise quadratic functions.

Figure 2.9: Function approximation using collocation methods [23].

Orthogonal collocation is a subset of direct collocation. It typically uses high-order splines, and each segment of the trajectory can be represented by a spline of a different order. The name comes from the use of orthogonal polynomials in the state and control splines.

To conclude, collocation methods create a large sparse non-linear problem, which tends to be easier to solve than the small dense programs produced by shooting methods. Shooting techniques are suitable for applications with simple control and few path constraints [28]. Multiple shooting approaches are often preferred over single shooting ones, unless the control is extremely simple or the initial guess is quite good. In contrast, collocation methods are more appropriate for applications where the dynamics and control must be computed accurately and the structure of the control trajectory is not known a priori [23].

2.3 Multi-Robot Trajectory Planning

Many multi-robot systems require the coordination of mobile agents when navigating complex environments. Thus, it is required to compute collision-free trajectories from an initial to a final position for every autonomous mobile robot in the fleet. Motion planning in multi-robot systems can be rather challenging since each robot must avoid both static and moving obstacles, in which the latter might be other vehicles in the fleet. Additionally, this becomes even more demanding when differential constraints are imposed by the physical dynamics and limited control inputs are considered. Also, it can be difficult to efficiently compute optimal collision-free routes, where the trajectory quality is determined by a cost function that, for example, minimizes the control effort or the trajectory duration. Many methods have been proposed for multi-robot trajectory planning. They can be divided into centralized, decentralized, and decoupled approaches.

In [29, 30], **centralized** techniques were used, where the optimal trajectory generation problem was formulated as Mixed Integer Quadratic Programming (MIQP) or Sequential Convex Programming (SQP) problems. The solutions generated with centralized methods are complete and optimal. Nevertheless, they are not scalable or robust.

To overcome these limitations, **decentralized** planning methods have been proposed. In this category are included the reactive methods such as Velocity Obstacle (VO) [31, 32] and buffered Voronoi cells [33]. Besides, in [34], was proposed a Distributed Model Predictive Control (DMPC). Distributed planning methods are computationally efficient, but these approaches do not support higher order dynamics and are relatively poor in certain scenarios, such as handling deadlocks.

Additionally, motion planning in multi-robot systems can be solved using **decoupled** techniques, such as sequential planning methods [35–37]. In sequential planning, the trajectory of each robot is decoupled and it is generated by avoiding the previously planned ones. This method is relatively fast, however, it can only find locally optimal solutions, not global ones.

In the context of this dissertation's motivation, it is crucial that the solution presented is robust to failures. In fact, one of the most important requirements of this project, which takes place in an industrial setting, is to ensure that the production line is correctly supplied and never interrupted. Therefore, since robots trajectories must be generated when asynchronous tasks are requested, this project follows a sequential planning approach.

2.4 Dynamic Map Update

A correct representation of the operating environment is essential for successful navigation of AMRs. Thus, it is necessary to guarantee that changes in the environment are detected and, posteriorly, shared between all vehicles. In multi-agent systems, this detection and updating must be done with cooperative perception, which is the principle of vehicles cooperating to extend their field of vision. This section starts by giving an overview on Map Merging techniques. Then, two relevant Map Sharing algorithms are presented.

2.4.1 Map Merging

A survey on map merging techniques can be found at [38]. Several significant types of maps, including occupancy gridmaps, feature-based maps, and topological maps, are examined in terms of their merging techniques.

Each cell in an occupancy gridmap represents the probability of occupation in that particular cell. Four methods for merging occupancy gridmaps are presented: Probabilistic, Optimization, Feature-based and Hough transform. The **probabilistic** approach was introduced in [39] and assumes that the robots know each other's initial pose. The **optimization** method designs an objective function that is dependent on the rigid transformation (rotation and translation) and maximizes the extent of coincidence between two maps. Nonetheless, both the probabilistic and the optimization approaches incur a significant computing cost. The **feature-based** technique extracts map features and merges occupancy gridmaps based on the larger number of common features. However, a significant shortcoming of this final strategy is the possibility of feature mismatching, when two identical features on distinct maps are misidentified as the same feature. Finally, Carpin [40] presented a transformation between maps based on the **Hough transform**. This algorithm is fast and robust, and, therefore, is suitable for real-time map merging.

Feature-based maps can reflect the environment more effectively than occupancy gridmaps. These maps focus on extracting features from the environment, and they can be classified into different types such as plane, line, point, etc. Identifying if numerous characteristics from distinct sources belong to the same structure is the primary downside of merging features. Merging **plane feature** maps offers a more robust option, since these features are less susceptible to noise. However, the computational cost on feature extraction is higher.

Finally, topological maps are less intuitive in terms of environment representation. Hence, there are less relevant works in the existing literature compared to those stated above. In topological maps, nodes are used to represent a set of features, while connecting edges represent possible paths between those features. Huang and Beevers [41] proposed a method that uses both map structure and map geometry to determine the best correspondence between maps with multiple overlapping regions. This approach proved to have a smaller computational cost. However, it can not be used to merge dynamic environments with moving obstacles.

2.4.2 Map sharing

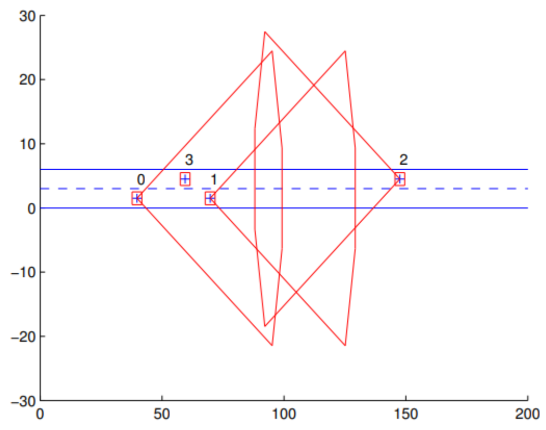
In [42] is proposed a map updating technique for multi-robot applications. Each robot owns an initial occupancy gridmap of the environment. The robots detect changes in the environment using a weighted recency averaging technique, creating a temporary map. Then, temporary maps are merged into the initial map by using Hough transform. Finally, the updated map is dispatched to the other robots, and each robot updates its own map based on the new information. Additionally, [43] uses an occupancy gridmap combined with a hidden Markov model that represents the belief about the occupancy and occupancy change probabilities for each cell in the grid, to compute an optimal roadmap.

In [44] a distributed fusion algorithm is used. Each vehicle detects a Local Dynamic Map (LDM) containing static and dynamic information about its immediate surroundings. Also, every vehicle has a confidence level m in the detected object's existence. A Dynamic Distributed Map (DDM) is a LDM that has been updated with maps received from other robots. By merging a DDM with a LDM of the robot, the Dynamic Public Map (DPM) is created, which is then sent to the network and shared with others.

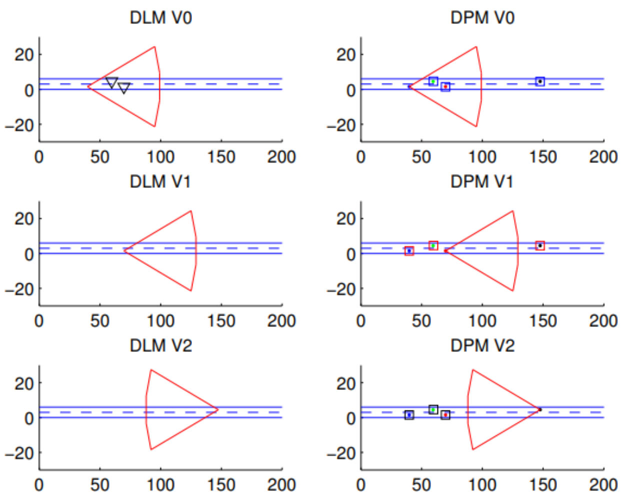
In Algorithm 5 the receiver gets the sender's DPM_s. The confidence m in DPM_s is reduced due to a discounting process, α. Additionally, the sender's position S is added to its map (line 2). Then, the DPM_s of the sender and the DDM_r of the receiver are predicted for some specific time, to obtain a temporal alignment (lines 3 and 4). The DDM_r is calculated by combining the predicted $\widehat{DPM}_s$ and $\widehat{DDM}_r$ with the confidence m of both maps (line 5). As a result of a similar merging process, the DPM_r is generated by combining the DDM_r and LDM_r, in which the latest map is built by the receiver robot sensors. The DPM_r is finally shared with the other robots.

Algorithm 5 Distributed Dynamic Map Algorithm

```
    Input: message from sender $DPM_s$
    Output: $DPM_r$, $DDM_r$
1:  Update $DDM_r$ with message
2:  $DPM_s \leftarrow^{\alpha} DPM_s + S$
3:  $\widehat{DPM}_s \leftarrow$ prediction($DPM_s$ )
4:  $\widehat{DDM}_r \leftarrow$ prediction($DDM_r$ )
5:  $DDM_r \leftarrow$ FusionCautious($\widehat{DPM}_s, \widehat{DDM}_r$)
6:  Compute $DPM_r$ with Local Map
7:  $LDM_r \leftarrow$ local map acquisition()
8:  $DPM_r \leftarrow$ FusionDempster($DDM_r, LDM_r$)
9:  Send $DPM_r$
```



(a) Vehicles V0, V1, V2 can exchange information. The goal is to detect V3. Each vehicle's field of view is highlighted in red.



(b) Dynamic Local Map of each vehicle (left column) and Dynamic Public Map of each vehicle (right column). Each vehicle's field of view is highlighted in red.

Figure 2.10: Results obtained with the Distributed Dynamic Map Algorithm [44].

Figures 2.10a-b depict the results of the Distributed Dynamic Map Algorithm presented above. Vehicles V_0, V_1 and V_2 can communicate with each other, while V_3 is the obstacle to be detected. Each vehicle's field of view is highlighted in red. The LDMs built with each vehicle sensors are shown in the first column of Figure 2.10b. Vehicle V_0 can detect V_1 and V_3. Vehicles V_1 and V_2 cannot detect anything inside their fields of view. Nevertheless, by sharing all this information, each vehicle can compute its own DPM. Therefore, all vehicles can have an updated map of the environment and be aware of the whereabouts of all robots and obstacle.

Chapter 3

Proposed Approach

The current chapter addresses the proposed methodology to generate each robot's optimized trajectory. Firstly, in Section 3.1, terms from the literature on motion planning are defined. Next, in Section 3.2 and Section 3.3, the mathematical problem formulation and the kinematic model of the vehicle under study are described, respectively. Finally, in Section 3.4, the proposed method for generating each robot's optimized trajectory are introduced.

3.1 Differential Constraints

In the previous chapter, the concepts of global and local constraints were introduced, which are imposed by obstacles and kinematic and dynamic considerations, respectively. Local constraints can also be denominated as differential constraints since they are modelled using differential equations. Consequently, the motion planning problem that considers both constraints should not only search for a trajectory in $\mathcal{C}_{free}$, but in a feasible configuration space, $\mathcal{C}_{feas}$. Nevertheless, it is important to define some terms from planning literature [9]:

- **Holonomic planning** - Refers to constraints that usually lead to a loss of the system's Degrees of Freedom (DoF). They can also be referred to as configuration constraints since they constraint the position of the agent.
- **Nonholonomic planning** - Refers to constraints modeled with differential equations. These constraints limit the space of possible velocities and can also be referred to as velocity or kinematic constraints.
- **Kinodynamic planning** - Refers to motion planning problems that satisfy velocity and acceleration bounds. More recently, the term has been applied to motion planning problems that take into account the agent's dynamics. Thus, kinodynamic planning can be considered as any problem that involves second-order (or higher) differential constraints.

3.2 Problem Formulation

Consider a multi-robot system consisting of N robots, $\mathcal{R} = \{R^1, ..., R^N\}$, that move in an Two Dimensional (2-D) environment, $\mathcal{W} \in \mathcal{R}^2$ populated by a set of M obstacles $\mathcal{O} = \{O^1, ..., O^M\}$. The set of all possible states of agent $i = 1, ..., N$ are defined by the state space $\mathcal{X}^i$, while the set of the allowable control input space is represented by $\mathcal{U}^i$.

The set of states that satisfies the global constraints are defined as $\mathcal{X}_{free}$. The dynamic constraints, that should be considered in this planning problem, are expressed in a differential state model or transition equation:

$$\dot{\mathbf{x}}(t) = f(t, \mathbf{x}(t), \mathbf{u}(t)), \tag{3.1}$$

defined for every state $\mathbf{x} \in \mathcal{X}$ and input $\mathbf{u} \in \mathcal{U}$.

For each robot R^i, a task s is assigned at time t^i_{init}. Every task should guarantee that the respective robot generates a feasible trajectory from its initial state, $\mathbf{x}^i_{init} \in \mathcal{X}^i$, to a goal state, $\mathbf{x}^i_{goal} \in \mathcal{X}^i$. Hence, the trajectory of robot R^i can be written as $\pi^i : [t^i_{init}, t^i_{goal}] \rightarrow \mathcal{X}_{free}$. Each trajectory is defined by: its duration, T^i; the control input along the trajectory, $\mathbf{u^i} : [t^i_{init}, t^i_{goal}] \rightarrow \mathcal{U}^i$; and the corresponding states, $\mathbf{x^i} : [t^i_{init} t^i_{goal}] \rightarrow \mathcal{X}^i$.

In a multi-robot system, given the workspace $\mathcal{W}$ and the tasks $\mathcal{S} = \{s^1, ..., s^K\}$ for the robot set $\mathcal{R}$, it is necessary to find a set of trajectories $\mathcal{P} = \{\pi^1, ..., \pi^K\}$ that execute the K tasks. Note that, any pair $\{\pi^i, \pi^j\}$ of every two different robots $\{R^i, R^j\}$ must be conflict free. Additionally, the tasks are assigned sequentially, i.e., the trajectory π^i is planned by avoiding the already defined ones: $\pi^1, ..., \pi^{i-1}$.

This formulation is represented in Figure 3.1. Three different motion query problems are illustrated, as well as three arbitrary paths that connect the initial to the goal states of each agent.

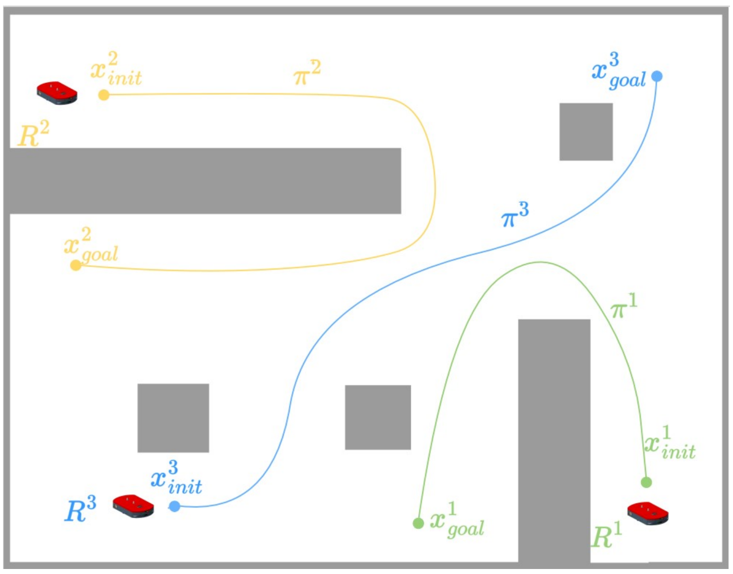
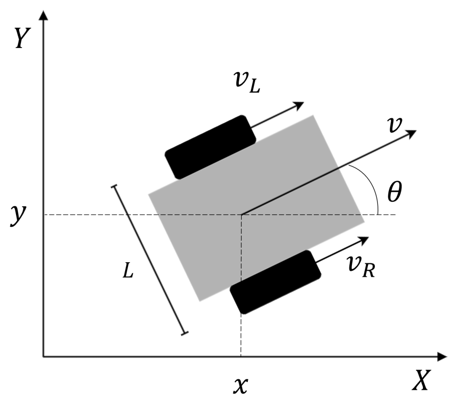


Figure 3.1: Motion planning problem for a fleet of three AMRs. For each vehicle R^i, $i = \{1, 2, 3\}$, an initial x^i_{init} and final x^i_{goal} configurations are assign to generate a trajectory π^i.

3.3 Kinematic Model of the Vehicle

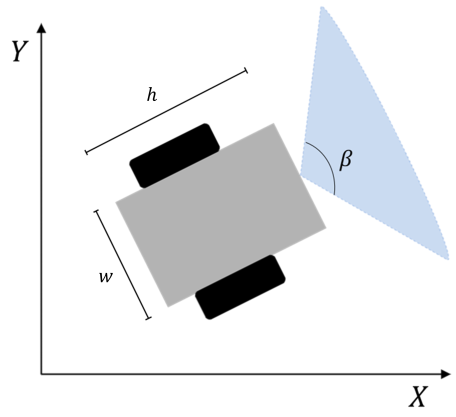
Without loss of generality, the proposed framework will be experimented on a fleet of differential drive robots. Notwithstanding, this method can be applied to any vehicle with kinodynamic constraints expressed as a differential state, as defined in Equation (3.1). Figure 3.2a depicts the kinematic model of a differential drive vehicle with two controllable wheels. This planar representation is appropriate since the modelling process is carried out according to a 2-D perspective of the vehicle's motion. The robot is equipped with sensors with an angular range β, as represented in Figure 3.2b. The vehicle's length and width are represented by h_r and w_r, respectively.

A differential drive vehicle has two independently actuated wheels on a common axis that are separated by a distance L. The vehicle's state vector can be expressed as $\mathbf{x} = [x_r, y_r, \theta_r, t, v_R, v_L]^T \in \mathcal{X}$, where x_r and y_r are the robot's cartesian coordinates, θ is the robot's orientation with respect to the x-axis, t denotes the time when each state is reached and, finally, v_R and v_L are the linear velocities of the right and left wheels, respectively.



o

(a) Vehicle parameters



(b) Sensor angular range

Figure 3.2: Vehicle representation.

The kinematic model of the system is given by the following equations of motion:

$$\begin{aligned}
\dot{x}_r &= v\cos\theta, \\
\dot{y}_r &= v\sin\theta, \\
\dot{\theta}_r &= w, \\
\dot{t} &= 1, \\
\dot{v}_R &= a_R, \\
\dot{v}_L &= a_L,
\end{aligned} \tag{3.2}$$

where v is the linear average velocity of the robot, given by:

$$v = \frac{v_R + v_L}{2}; \tag{3.3}$$

ω the angular velocity of the agent given by:

$$\omega = \frac{v_R - v_L}{L}; \tag{3.4}$$

and a_R and a_L represent the linear acceleration of the right and left wheels, respectively. Thus, the control vector can be defined as $\mathbf{u} = [v, \omega, a_R, a_L]^T$.

3.3.1 Reachable states of the vehicle

To obtain the reachable states of the vehicle, it is necessary to consider the limitations of the system. Note that both right and left wheels are subject to limits in acceleration and velocity. As mentioned previously, the robot is equipped with sensors with an angular range β. It should be guaranteed that the vehicle only moves within this range, otherwise, there is a high risk of colliding with obstacles. Therefore, some constraints on the vehicle's linear and angular velocities must be taken into account when generating the feasible states of the agent:

- Both wheels velocities and accelerations are bounded:

$$\begin{aligned} &|a_R| < a_{max}, \\ &|a_L| < a_{max}, \\ &|v_R| < v_{max}, \\ &|v_L| < v_{max}. \end{aligned} \tag{3.5}$$

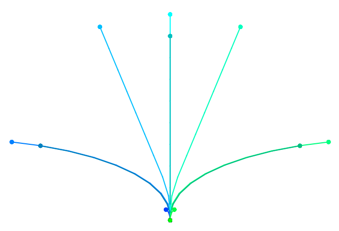
- The linear velocity of the robot cannot be a negative value, since the robot cannot perform a reverse maneuver:

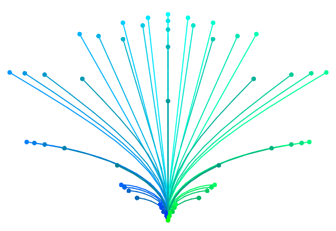
$$v > 0. \tag{3.6}$$

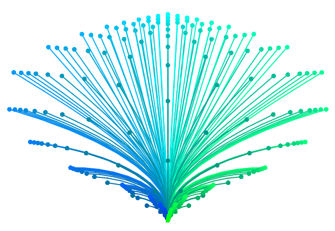
- The angular variation of the vehicle in each time step, dt, should insure that the robot only moves within the angular range of the sensor:

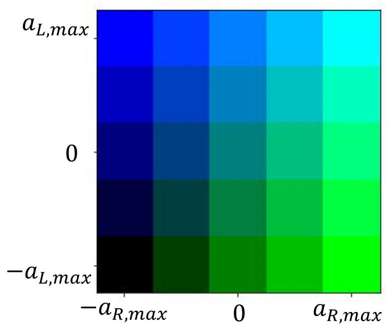
$$|w| < \frac{\beta/2}{dt}. \tag{3.7}$$

In fact, the bounded right and left wheel accelerations of the robot are both discretized to obtain the respective wheel velocities. With all the combinations between both wheel velocities, the average linear and angular velocities are mapped and used to generate all the possible attainable states after one time step. Figures 3.3a-c depict the attainable states of the robot for different numbers of discretized control actions, when the vehicle is initially at rest. Figures 3.3d-f show the respective colormap of the discretized accelerations. Each color represents a pair of right and left wheel velocities, that corresponds to the same color trajectory for the respective figure illustrating the reachable states. As an example, if on the left wheel is the maximum and the acceleration on the right wheel is the minimum (upper left values of the colormap), the vehicle will turn to the left. When the accelerations of both wheels are equal (diagonal values of the colormap), their velocities are also identical and the vehicle moves in a straight line.

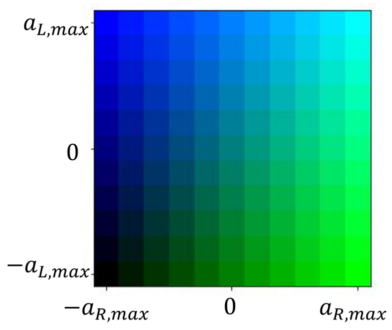

(a) Reachable states of the vehicle when both wheels acceleration is discretized 5 times.


(b) Reachable states of the vehicle when both wheels acceleration is discretized 11 times.

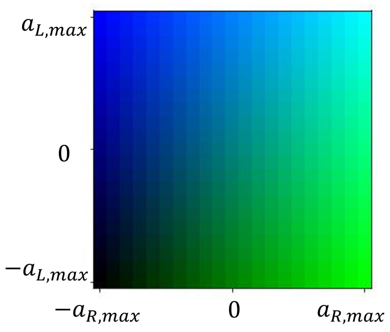

(c) Reachable states of the vehicle when both wheels acceleration is discretized 5 times.



(d) Right and left wheels accelaration discretized 5 times. Each color represents a pair of right and left wheel accelerations, that corresponds to the same color trajectory shown in (a).



(e) Right and left wheels accelaration discretized 11 times. Each color represents a pair of right and left wheel accelerations, that corresponds to the same color trajectory shown in (b).



(f) Right and left wheels accelaration discretized 21 times. Each color represents a pair of right and left wheel accelerations, that corresponds to the same color trajectory shown in (c).

Figure 3.3: Reachable states of the robot after one time step, when the vehicle starts at rest. Right wheel acceleration and left wheel acceleration are discretized 5 (a) and (d), 11 (b) and (e) and 21 (c) and (f) times.

3.4 Methodology

Most path planning algorithms described in Subsection 2.1, are applicable to holonomic systems, since they rely on the ability to connect any pair of states, not taking into account local (differential) constraints. For kinodynamic systems, straight-line connections between pairs of states are usually not valid trajectories. Thus, in this framework is necessary to find a feasible path, executable for the robot, and not only a collision-free one.

One common approach to solve the kinodynamic trajectory planning problem is to divide it into three sequential steps: path planning, trajectory optimization, and velocity planning [9]. The first step consists of computing a collision-free geometrical path from an initial to final configurations. Then, the path previously generated is processed and smoothed, having into consideration the kinematic limitations. In the last step is designed a velocity profile to execute the smoothed trajectory. Nevertheless, as illustrated in Figure 3.4, the use of a classical motion planning strategy has shown to be problematic for vehicles with differential constraints. Zhang et al. [45], acknowledge the importance of generating a good initial trajectory, to further be optimized and smoothed. An initialization that generates a path that does not acknowledge the system kinematics is, in general, infeasible and potentially not continuously transformable into a feasible solution. If the initial guess is an infeasible path, it is very often impossible for a solver to converge to a solution.

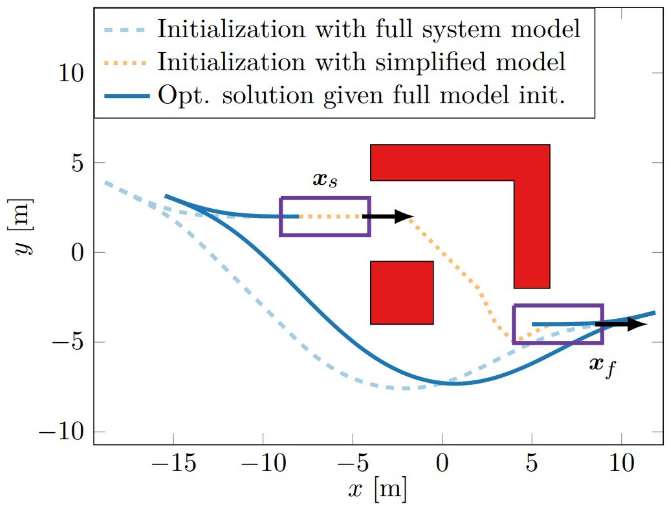


Figure 3.4: In dashed yellow is represented a solution from a path planner that disregards the system's dynamics completely. Therefore, in this situation is impossible for a solver to find a feasible trajectory. In comparison, the dash blue line represents a path generated by a path planning method that considers the system model, enabling to reach an optimal solution, represented by the blue line, when optimizing the initial trajectory [46].

The overall architecture of the proposed multi-robot motion planning approach is shown in Figure 3.5. To generate robot's R^i optimized trajectory is necessary to have information regarding the updated map of the environment $\mathcal{W}$, the already assigned trajectories of all vehicles $\mathcal{P}$ and the task s^i that is going to be assigned. This framework is divided into two modules: Trajectory Planning and Trajectory Optimization. In the first one, a sampling-based method is used to generate an initial trajectory that satisfies the task assigned to R^i. This module is discussed in Chapter 4. Then, in the Trajectory Optimization module, the trajectory computed previously is refined into a smoother and shorter one by solving an optimization problem. This is discussed in Chapters 5 and 6. Both modules directly consider the non-linear motion model of the robot, to guarantee the feasibility of the trajectories.

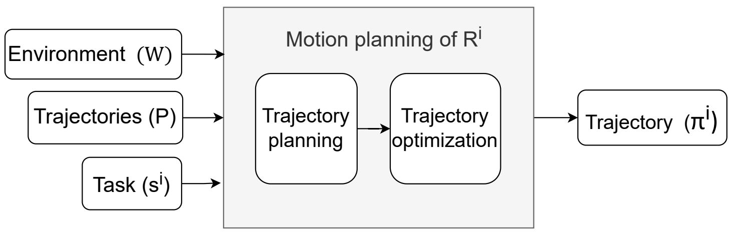


Figure 3.5: Architecture of the motion planning approach of robot R^i. This framework takes as input the updated map of the environment $\mathcal{W}$, the already assigned trajectories of all vehicles $\mathcal{P}$ and the task s^i that is going to be assigned. To generate an optimized trajectory π^i, two stages are considered: the Trajectory Planning and Trajectory Optimization.

Chapter 4

Single-Robot Trajectory Planning

In the present chapter, both global and local constraints are considered when searching for a feasible initial trajectory. The randomized method, which was conceptually described in Subsection 2.1, can be applied to the kinodynamic planning problem. In order to solve it, an extension of the RRT algorithm, the Kinodynamic RRT, assessed by La Valle [47], is explored, implemented and analyzed. In Section 4.1 the algorithm is presented and in 4.2 the Kinodynamic RRT is tested for a differential drive vehicle.

4.1 Kinodynamic RRT

When selecting a motion planner to use in this dissertation, the chosen algorithm must be complete and have a low complexity. The optimality of the method was not a big deciding factor since the trajectory is posteriorly optimized considering a cost function that depends on parameters to be defined. Additionally, since the RRT method, presented in Algorithm 2, can be extended to a kinodynamic system, it was suitable to be implemented in this framework. The Kinodynamic RRT computes not only a path, but a trajectory that considers the vehicle's kinematics and dynamics.

The Kinodynamic RRT begins by constructing a tree of trajectories $\mathcal{T} = (N, E)$ rooted in the initial state and searches for the goal region $\mathbf{X}_{goal}$, which is centered on the goal state. Again, the free space robot configurations are represented by the tree's nodes, N. Edges, E, can join two nodes if and only if the corresponding nodes can be connected by a feasible trajectory. The Kinodynamic RRT pseudocode is depicted in Algorithm 6. The creation and expansion of the $\mathcal{T}$ is illustrated in Figures 4.1a-g.

The algorithm starts by adding $\mathbf{x}_{init}$ to the tree $\mathcal{T}$ (line 1, Figure 4.1a).The *Steer* function computes a set $\mathbf{X}_{reach}$ (line 2). $\mathbf{X}_{reach}$ can be defined as the set of all states that can be reached from the *Steer* function input, $\mathbf{x}_{init}$, within some defined time step (Figure 4.1b). In order to do that, all the possible input controls are used, taking into consideration the limits on the discretized accelerations and velocities of the vehicle. A collision avoidance module is also required to determine whether or not configurations from set $\mathbf{X}_{reach}$ are collision-free (line 4). The states that do not collide with any obstacles can then be added to $\mathcal{T}$ (lines 5 and 6).

Each iteration of the Kinodynamic RRT samples a new random state from the entire state space $\mathcal{X}$. As discussed in [15], by returning occasionally the center of $\mathbf{X}_{goal}$ in the *BiasRandomState* function (line 10), with a probability defined by a goal bias value, allows the tree to converge to the goal region much faster. This goal-biased RRT was implemented in this project, bearing in mind the fact that by over considering the bias, the algorithm might get trapped in a local minima.

The node in $\mathcal{T}$ that minimizes the euclidean distance to $\mathbf{x}_{rand}$ is defined as $\mathbf{x}_{near}$ (line 11). Figure 4.1c illustrates nodes $\mathbf{x}_{rand}$ and $\mathbf{x}_{near}$. Then, the *Steer* function is used once again to generate a set of possible reachable states rooted in $\mathbf{x}_{near}$ (line 12). The collision-free configurations from this new set $\mathbf{X}_{near}$ are added to $\mathcal{T}$ (lines 15 and 16, Figure 4.1d).

Finally, in function *ReachGoalRegion* is inspected if any node in $\mathcal{T}$ has reached the goal region (line 19). In that case, the search is completed and the algorithm returns the final tree $\mathcal{T}$. Otherwise, the algorithm continues with a new iteration. If the number of iterations exceeds a predefined maximum value, the search is terminated and the algorithm fails to generate a trajectory. Figures 4.1f-g illustrate $\mathcal{T}$ reaching the goal region, $\mathbf{X}_{goal}$, and the final trajectory computed, respectively.

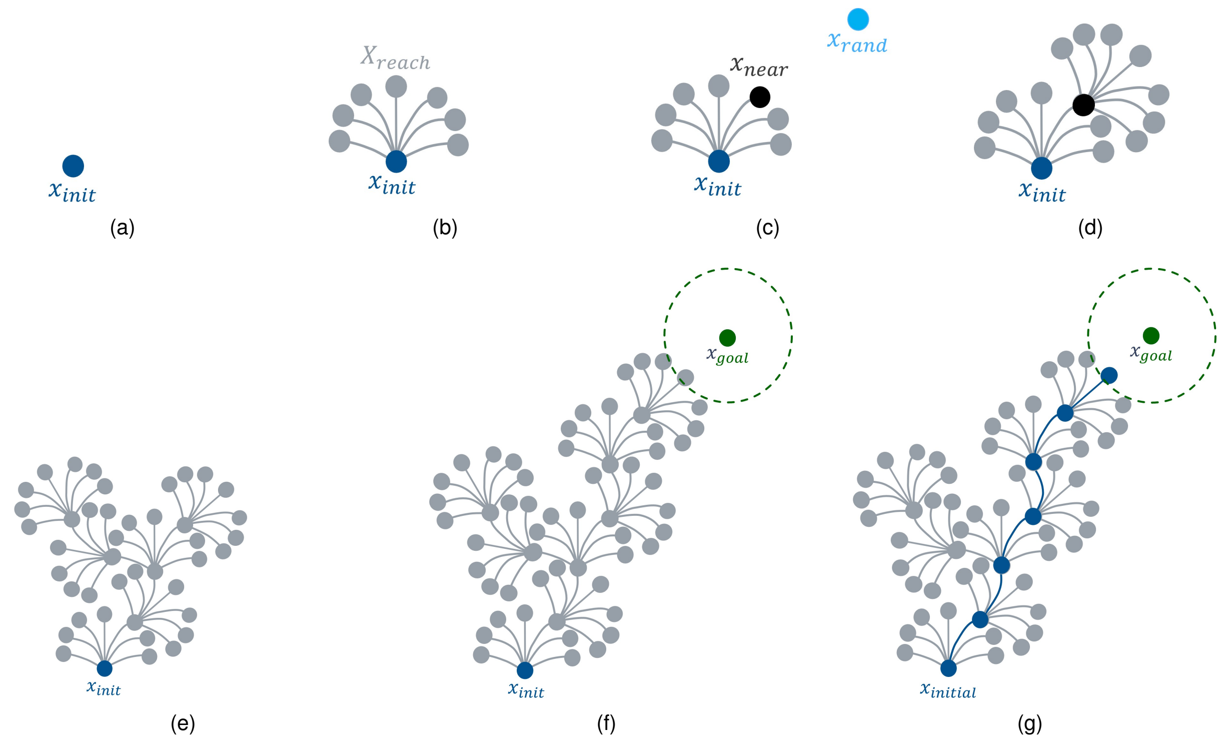


Figure 4.1: Construction of a Kinodynamic RRT tree, $\mathcal{T}$. First, the initial state $\mathbf{x}_{init}$ (dark blue) is added to $\mathcal{T}$ (a). Then the collision-free states, $\mathbf{X}_{reach}$ (light grey), that can be reached from $\mathbf{x}_{init}$ are included in $\mathcal{T}$ (b). A random state, $\mathbf{x}_{rand}$ (light blue), is sampled (c). $\mathbf{x}_{near}$ (black) is defined as the closest node in $\mathcal{T}$ to $\mathbf{x}_{rand}$ (c). Then, the set of reachable states rooted in $\mathbf{x}_{near}$ are added to $\mathcal{T}$ (d). The tree $\mathcal{T}$ grows (e) until the goal region $\mathbf{X}_{goal}$ (green dashed circle) is reached (f). Finally, the node that reached $\mathbf{X}_{goal}$ is connected to $\mathbf{x}_{init}$ generating a trajectory (g).

Algorithm 6 Kinodynamic RRT

Input: initial state $\mathbf{x}_{init}$

Output: kinodynamic RRT tree $\mathcal{T} = (N, E)$

```
1:  $N \leftarrow \{\mathbf{x}_{init}\}$
2:  $\mathbf{X}_{reach} \leftarrow$ Steer($\mathbf{x}_{init}$)
3:  for $\mathbf{x} \in \mathbf{X}_{reach}$ do
4:      if CollisionFree($\mathbf{x}_{init}, \mathbf{x}$) then
5:          $N \leftarrow N \cup \{\mathbf{x}\}$
6:          $E \leftarrow E \cup \{(\mathbf{x}_{init}, \mathbf{x})\}$
7:      end if
8:  end for
9:  for $i < max\_iter$ do
10:     $\mathbf{x}_{rand} \leftarrow$ BiasRandomState($\mathcal{X}$)
11:     $\mathbf{x}_{near} \leftarrow$ NearestNode($\mathbf{x}_{rand}, \mathcal{T}$)
12:     $\mathbf{X}_{reach} \leftarrow$ Steer($\mathbf{x}_{near}$)
13:     for $\mathbf{x} \in \mathbf{X}_{near}$ do
14:         if CollisionFree($\mathbf{x}_{near}, \mathbf{x}$) then
15:             $N \leftarrow N \cup \{\mathbf{x}\}$
16:             $E \leftarrow E \cup \{(\mathbf{x}_{near}, \mathbf{x})\}$
17:         end if
18:     end for
19:     if ReachGoalRegion($\mathcal{T}$) then
20:         return $\mathcal{T}$
21:     end if
22: end for
23: return Failure
```

4.2 Trajectory Planning with a Differential Drive Robot

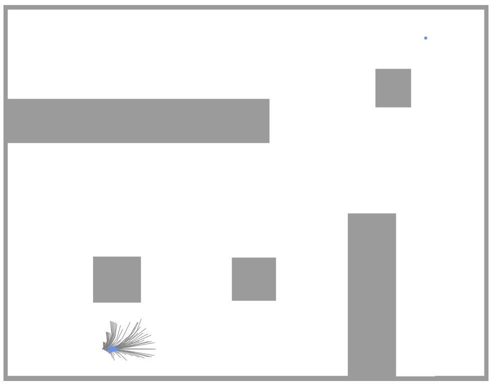
The following results are depicted in a $25m \times 20m$ environment with 5 obstacles. The differential drive vehicle has dimensions $h_r = 1.2m$ and $w_r = 0.72m$. The distance between both wheels of the robot is $L = 0.63m$ and the sensor has an angular range of $\beta = 180^\circ$. The results were obtained considering a maximum velocity and acceleration for each wheel of $v_{max} = 1ms^{-1}$ and $a_{max} = 0.5m^2s^{-1}$, respectively.

Figures 4.2a-c illustrate the construction process of a single tree using the proposed algorithm. The Kinodynamic RRT quickly expands towards the goal region. Figure 4.2d depicts the final trajectory obtained after 55 iterations. It can be verified that the trajectory connects the initial state and to the goal region.

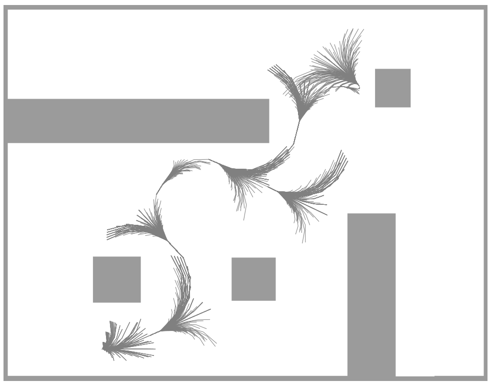
Finally, to show the random property of the Kinodynamic RRT, the algorithm was run fifty times and, as expected, fifty different trajectories were generated for the same motion planning problem. Figure 4.3a reports the trajectories obtained. The data collected is depicted in Figure 4.3b, using violin representation. Violin plots combine a box plot with a kernel density estimate. In this example the vertical axis reflects the data being analysed, while the horizontal axis indicates the kernel density plot. This representation describes the entire data range as well as the mean, minimum, and maximum values of the data. The kernel density plot provides the probability density function of the studied variable.

The violin plots in Figure 4.3b depict the time required to execute the generated trajectory, i.e., the trajectory cost, as well as the algorithm's computational time [1]. Lastly, the third plot illustrates the

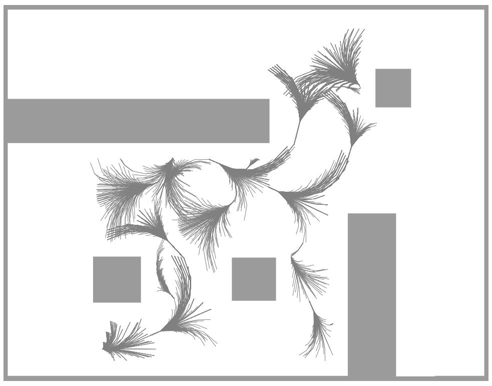
[1]The simulations were performed on a desktop with an Intel Core i7 U-Processor Processor 1.80GHz, 16GB of RAM and Intel(R) UHD Graphics. The operative system used was Microsoft Windows 11 Pro.



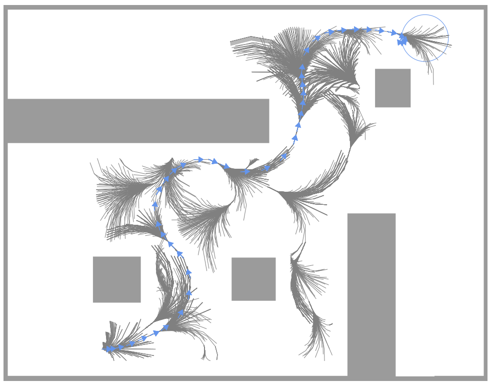
(a) Initial state (light blue) and reachable states of the vehicle (grey) after 1 iteration of the Kinodynamic RRT algorithm.



(b) Reachable states of the vehicle (grey) after 20 iterations.



(c) Reachable states of the vehicle (grey) after 40 iterations.

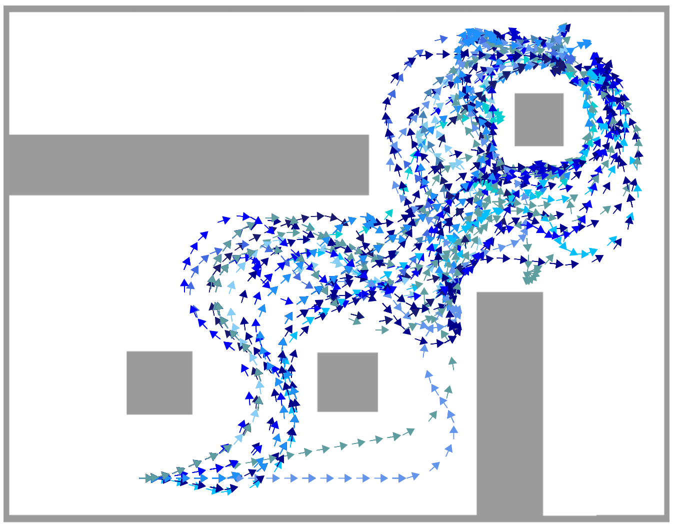


(d) Reachable states of the vehicle (grey) after 55 iterations. The goal region (light blue circle) is reached and a trajectory (light blue arrows) is computed.

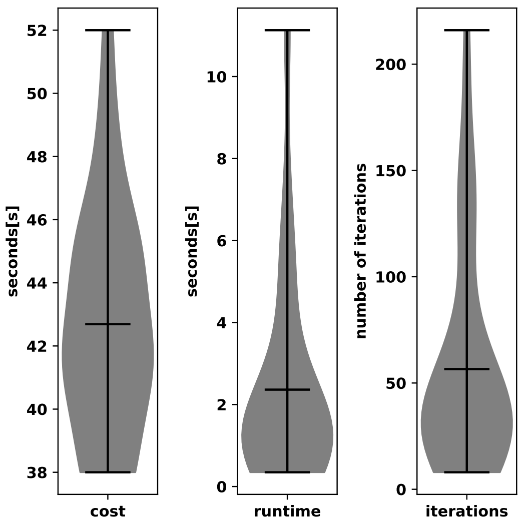
Figure 4.2: Single tree construction after 1 (a), 20 (b), 40 (c) and 55 (d) iterations.

amount of iterations required to generate one trajectory.

The results reveal that the algorithm implemented can create a collision-free and feasible trajectory in the given environment. It fully explores the differential drive vehicle manoeuvring ability while considering its kinematics. Nevertheless, some drawbacks should be highlighted. The low quality trajectory is a direct result of the algorithm's inherent randomness. In fact, the solution does not follow a smooth trajectory and may contain unnecessary manoeuvrers to achieve the goal region. As expected, the randomized-based approach was insufficient to guarantee the generation of optimized trajectories that the vehicle can execute. The provided techniques are good at delivering an initial solution, but they must be coupled with more sophisticated methods in order to provide higher quality routes. In the following chapter, a trajectory optimization method is developed to shorten and smooth out the obtained Kinodynamic RRT trajectories.



(a) Different trajectories generated for the same motion problem with the Kinodynamic RRT trajectory planner. Each arrow represents a vehicle's pose. These results depict the inherent randomness of the algorithm.



(b) Data distribution regarding trajectory duration, computational time and number of iterations to generate one trajectory with the Kinodynamic RRT planner. The violin plots depicts the probability density function of the studied variable. Additionally, the lines visually represent the mean value of the data, while the whiskers delimit the minimum and maximum data points.

Figure 4.3: Fifty different trajectories generated using the Kinodynamic RRT for the same motion problem (a). The violin plots display the data distribution regarding trajectory duration, computational time and number of iterations to generate one trajectory (b).

Chapter 5

Single-Robot Trajectory Optimization

This chapter formulates an optimization problem that can be used to smooth and enhance, according to some cost function, the trajectory provided by the Kinodynamic RRT. Section 5.1 focuses on describing the constrained trajectory optimization problem through differential equations and algebraic equalities and inequalities. In Section 5.2 the optimization problem is validated for a differential drive vehicle.

5.1 Optimization Problem

When deciding on a trajectory optimization technique, it is important to find a compromise between the computation complexity of the method and its solution's accuracy. In an industrial environment is not crucial that the computed routes are not time-continuously defined. One the other hand, high computational times can be harmful to the multi-system coordination. Therefore, discrete approaches were taken into account, such as direct methods, since they are less difficult to construct and solve. Direct methods discretize the trajectory optimization problem and convert it into a non-linear one. Also, the Kinodynamic RRT trajectory is already represented in a discrete-time system, since each discrete node contains information regarding the vehicle's pose at that specific time instant. Therefore, the initial guess solution provided by the Kinodynamic RRT is already prepared to be used by this approach. Inside the family of direct methods, direct collocation techniques can deal with more complex dynamics and control systems. In this dissertation, the trajectory optimization problem will be formulated using the trapezoidal collocation method.

With trapezoidal quadrature, the discrete problem is represented by a set of collocation constraints [27]. This is accomplished by representing the continuous state $\mathbf{x}(t)$, control $\mathbf{u}(t)$ and system dynamics $\mathbf{f}(t, \mathbf{x}(t), \mathbf{u}(t))$ as their values at specific time points known as collocation points. For reference, the main symbols used throughout the following chapters can be defined as follows:

- t_k: time at knot point $k \in \{0, ..., C-1\}$;
- C: number of collocation points;

- $h = t_{k+1} - t_k$: time interval between consecutive collocation points;
- $\mathbf{x}_k = \mathbf{x}(t_k)$: state at knot point $k \in \{0, ..., C-1\}$;
- $\mathbf{u}_k = \mathbf{u}(t_k)$: control at knot point $k \in \{0, ..., C-1\}$; and
- $\mathbf{f}_k = \mathbf{f}(t_k, \mathbf{x}_k, \mathbf{u}_k)$: systems dynamics at knot point $k \in \{0, ..., C-1\}$.

In this section, the dynamic constraints are presented, along with the state and control bounds, the boundary constraints and the collision avoidance constraints. Lastly, an overall framework for the formulated optimization problem is provided.

Dynamic Function

In direct collocation methods, the differential state model can be considered as a constraint on the dynamics of the vehicle. By integrating both sides of the differential state defined in Equation (3.1), it is possible to get the change in state between two consecutive collocation points:

$$\int_{t_k}^{t_{k+1}} \dot{\mathbf{x}}(t)dt = \int_{t_k}^{t_{k+1}} \mathbf{f}(t, \mathbf{x}(t), \mathbf{u}(t))dt. \tag{5.1}$$

This integral can be approximated using trapezoidal quadrature:

$$\mathbf{x}_{k+1} - \mathbf{x}_k \approx \frac{1}{2}h\left(\mathbf{f}_k + \mathbf{f}_{k+1}\right), \tag{5.2}$$

with $k \in \{0, ..., C-1\}$. This approximation is then applied between every pair of collocation points, where the discrete time nodes along the trajectory are defined as $t_k = kh$.

State and Control bounds

The physical limits of the system must be considered when defining the optimization problem. These constraints are applied throughout the entire trajectory to the state and control variables. They are given by:

$$\mathbf{x}_{min} \leq \mathbf{x}_k \leq \mathbf{x}_{max}, \tag{5.3}$$

$$\mathbf{u}_{min} \leq \mathbf{u}_k \leq \mathbf{u}_{max}, \tag{5.4}$$

with $k \in \{0, ..., C-1\}$.

Boundary Constraints

Boundary constraints satisfy the requirements of the initial and final states. They are given by some function:

$$g(t_{init}, t_{goal}, \mathbf{x}_0, \mathbf{x}_{C-1}) \leq 0. \tag{5.5}$$

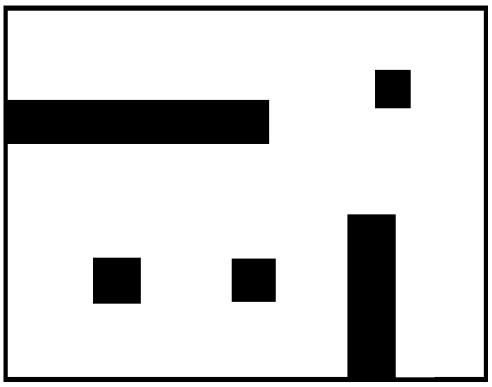
Collision Avoidance Constraint

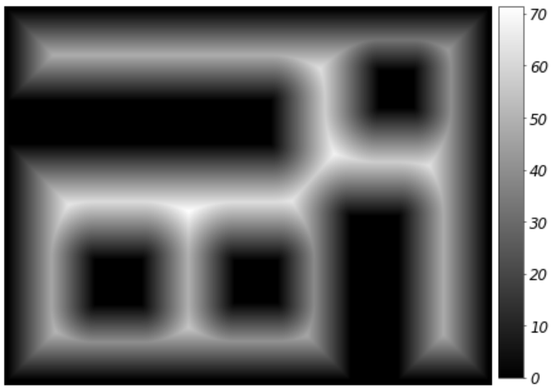
Let $\mathbf{p}(t) \in \mathbf{x}(t)$ be the cartesian coordinates of a robot R for all $t \in [t_{init}, t_{goal}]$. A trajectory is considered collision-free if the distance between each vehicle position along the trajectory and the nearest obstacle, $\mathbf{p}(t)$, is greater than a safe margin, $d_{safe} > 0$.

When dealing with static obstacles, it is preferable to precompute a Signed Distance Field (SDF) that stores the signed distance from any point $\mathbf{p}(t)$ to the nearest obstacle's boundary. Values for the signed distance are negative inside obstacles, positive outside and zero at the boundary [48]. For a collision-free trajectory, it is necessary to ensure at every position of R:

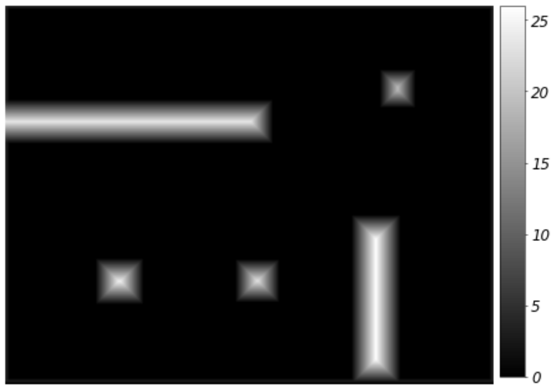
$$sd(\mathbf{p}(t), O^m) \geq d_{safe}, \quad \forall m \in \{1, ..., M\}. \tag{5.6}$$

The pseudocode for computing a SDF from a binary occupancy grid map is depicted in Algorithm 7. Fig 5.1a illustrates an example of an occupancy gridmap, where black represents the obstacles and white represents unoccupied areas. The EDT for both the occupancy gridmap and its complement is computed, as presented in lines 1-2 and Figures 5.1b-c, respectively. The SDF is then given by the

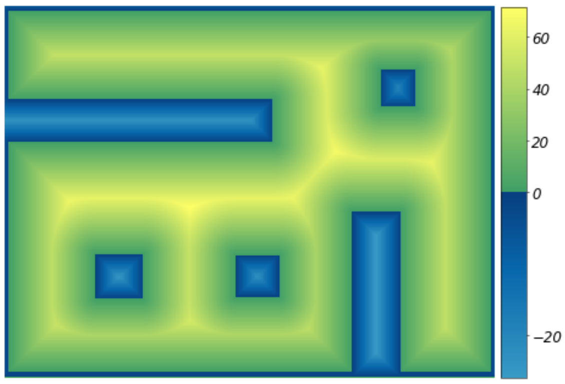

(a) Occupancy gridmap (G). Black represents the obstacles and white the unoccupied areas.



(b) Euclidean distance transform of the occupancy gridmap (EDT_G). Depicts the distance to the nearest obstacle.



(c) Euclidean distance transform of the occupancy gridmap logical complement ($EDT_{\overline{G}}$). Depicts the distance to the nearest unnocupied area.



(d) Signed distance field (SDF). Green and yellow represent areas outside the obstacles, and blue are areas inside the object.

Figure 5.1: Example of occupancy gridmap and its respective Euclidean Distance Transform (EDT)s and SDF.

difference between the two EDT values (line 3). Figure 5.1d presents the SDF computed from the known map, where the darker blue represents the obstacle boundaries, the light blue the area inside the objects, and green and yellow the areas outside the obstacles (yellow being the furthest from an obstacle).

Algorithm 7 Computation of a Signed Distance Field

```
    Input: binary occupancy gridmap $G$
    Output: signed distance field $SDF$
1:  $EDT_G \leftarrow EuclideanDistanceTransform(G)$
2:  $EDT_{\overline{G}} \leftarrow EuclideanDistanceTransform(\overline{G})$
3:  $SDF \leftarrow EDT_G - EDT_{\overline{G}}$
4:  return $SDF$
```

Note that a SDF is described by a discrete grid system. For it to be applicable to the optimization problem, the SDF is interpolated to obtain a differentiable function. This function takes a robot's pose $\mathbf{p}(t)$ as an input and returns the signed distance from $\mathbf{p}(t)$ to the nearest static obstacle. This function can be defined as:

$$d(\mathbf{p}(t)). \tag{5.7}$$

Cost Function

A common used cost function is the integral of control effort squared. This cost function is desirable because it computes smooth solution trajectories [23]. This objective function can be approximated using trapezoidal quadrature, converting it into a sum over each segment, using the control error at every collocation point:

$$J(h, \mathbf{x}_k, \mathbf{u}_k) = h\frac{1}{2}\sum_{k=0}^{C-1}(\mathbf{u}_k^2 + \mathbf{u}_{k+1}^2). \tag{5.8}$$

Without loss of generality, in this dissertation, a cost function was chosen to minimize the total duration of the vehicle's trajectory. By using direct collocation, this can be achieved by minimizing the time step, i.e:

$$J(h, \mathbf{x}_k, \mathbf{u}_k) = h. \tag{5.9}$$

Additionally, in Chapter 7, a reformulated cost function that attracts the vehicles to drive on the right side of the factory's corridors is presented .

Proposed Formulation

Finally, the overall trajectory optimization problem of robot R^i can be defined as follow:

$$
\begin{aligned}
\text{Minimize} \quad & J(h, \mathbf{x}_k, \mathbf{u}_k) = h && (5.10)\\
\text{w.r.t.} \quad & h, \mathbf{x}_k, \mathbf{u}_k, \quad \forall k \in \{0, ...C-1\} \\
\text{subject to} \quad & \mathbf{x}_{k+1} - \mathbf{x}_k = \frac{1}{2}h(f_k + f_{k+1}), \quad \forall k && (5.11)\\
& d(\mathbf{p}(hk)) \geq d_{safe}, \quad \forall k && (5.12)\\
& g(t_{init}, t_{goal}, \mathbf{x}_0, \mathbf{x}_{C-1}) \leq 0 && (5.13)\\
& \mathbf{x}_{min} \leq \mathbf{x}_k \leq \mathbf{x}_{max}, \quad \forall k && (5.14)\\
& \mathbf{u}_{min} \leq \mathbf{u}_k \leq \mathbf{u}_{max}, \quad \forall k && (5.15)
\end{aligned}
$$

Initial Guess

It is critical to have a good initial guess when solving a trajectory optimization problem, as this allows the solver to quickly arrive at the optimal solution. In this dissertation, the trajectory generated by the Kinodynamic RRT, as specified in Section 4, serves as the initial guess.

5.2 Trajectory Optimization with a Differential Drive Robot

The results presented in this section were obtained using the differential drive model described in Chapter 3. Using the non-linear equations of motion presented in (3.2) and the trapezoidal quadrature approximation in (5.2), the expanded collocation constraints for the kinematic model can be defined as:

$$
\begin{aligned}
x_{r,k+1} - x_{r,k} &= \frac{1}{2}h\left(v_k cos(\theta_k) + v_{k+1} cos(\theta_{k+1})\right), \\
y_{r,k+1} - y_{r,k} &= \frac{1}{2}h\left(v_k sin(\theta_k) + v_{k+1} sin(\theta_{k+1})\right), \\
\theta_{r,k+1} - \theta_{r,k} &= \frac{1}{2}h\left(\omega_k + w_{k+1}\right), \\
v_{R,k+1} - v_{R,k} &= \frac{1}{2}h\left(a_{R,k} + a_{R,k+1}\right), \\
v_{L,k+1} - v_{L,k} &= \frac{1}{2}h\left(a_{L,k} + a_{L,k+1}\right),
\end{aligned}
\tag{5.16}
$$

with $k \in \{0, ..., C-1\}$.

The state and control bounds of the vehicle under study, such as velocity and acceleration bounds, and angular variation, are presented in (3.5) - (3.7). These continuous inequalities are discretized into a set of constraints applicable to all collocation points:

$$\begin{gathered}
|v_{R,k}|\leq v_{max} \wedge |v_{L,k}|\leq v_{max},\\
|a_{R,k}|\leq a_{max} \wedge |a_{L,k}|\leq a_{max},\\
v_k \geq 0,\\
|\omega_k| \geq \frac{\beta/2}{h},
\end{gathered} \tag{5.17}$$

for all $k \in \{0, ..., C-1\}$.

The generalized boundary constraints in (5.5) can be modified for the specific case in which the vehicle initiates and finishes an assigned task at rest:

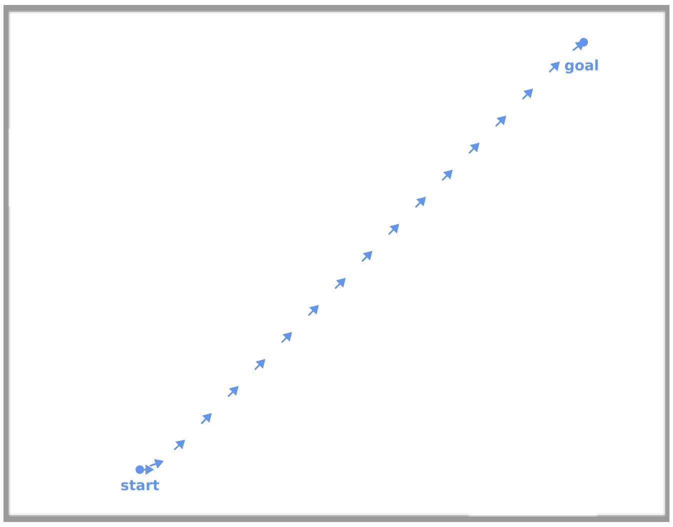
$$\begin{aligned}
\mathbf{x}_0 &= \left[x_{r,init}, y_{r,init}, \theta_{r,init}, 0, 0\right]^T,\\
\mathbf{x}_{C-1} &= \left[x_{r,goal}, y_{r,goal}, \theta_{r,goal}, 0, 0\right]^T.
\end{aligned} \tag{5.18}$$

5.2.1 Optimization Solver Integration

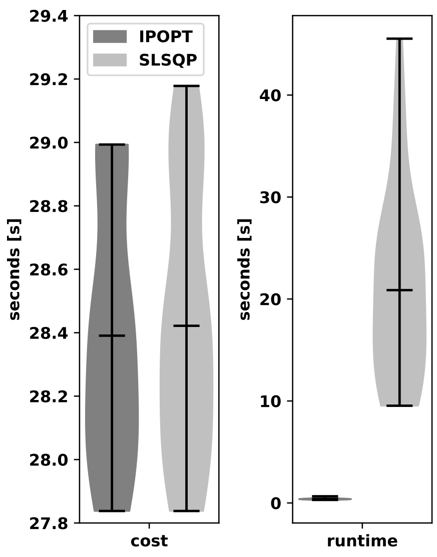
Two different optimizers were tested to solve the proposed optimization problem. The Sequential Least Squares Programming (SLSQP) [49, 50] approaches the problem as a sequence of constrained least-squares problems. The algorithm optimizes successive second-order approximations of the objective function with first-order approximations of the constraints. On the other hand, the Interior Point Optimizer (IPOPT) [51] uses an interior-point method to find local optimum solutions to large-scale non-linear optimization problems. Interior-point methods are also called barrier methods since they use barrier functions, which are continuous functions whose values go to infinity as the point gets closer to the edge of the feasible region.

The SLSQP was implemented using the open-source Python library SciPy [52] and the IPOPT solver using Python's symbolic framework CasADi [53]. The same motion planning problem was solved fifty times using both SLSQP and IPOPT, having for each iteration a different initial guess trajectory provided by the Kinodynamic RRT. The solution of an optimized trajectory that connects an initial to a goal pose is illustrated in Figure 5.2a. These tests on both optimizers were made in an obstacle-free scenario. In Figure 5.2b is depicted the gathered data regarding the cost function and the computational time. The value of the cost function corresponds to the time that takes the vehicle to execute the optimized trajectory. The computational time corresponds to the time that the solver takes to optimize the trajectory.

Both solvers optimized trajectories have approximately similar durations. Regarding the computational time, it can be verified that the IPOPT optimizer outperforms the SLSQP. With IPOPT it is possible to locally optimize this specific trajectory in less than 0.66 seconds. Further experiments were also conducted in an environment populated with obstacles and the IPOPT yield better computational results compared to the SLSQP. Therefore, the following tests presented in this dissertation will be implemented using as solver the IPOPT.



(a) Solution of optimized trajectory.

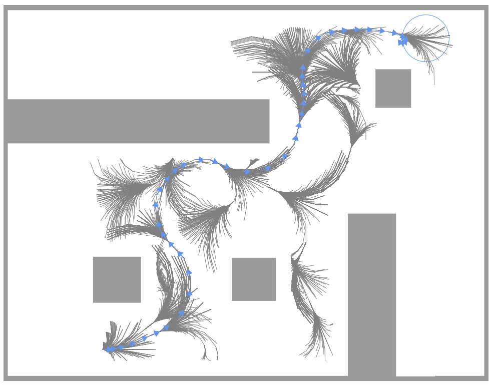


(b) Data distribution regarding trajectory duration and computational time for the IPOPT and SLSQP optimizers. The violin plot symbolism used is described in Figure 4.3b.

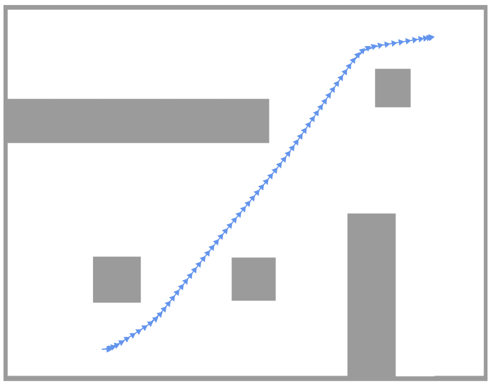
Figure 5.2: Comparison between two optimizers: SLSQP and IPOPT.

5.2.2 Single-Robot Results

The trajectory generated by the motion planning method described previously, the Kinodynamic RRT, is depicted in Figures 5.3a. Its generation is described in Chapter 4 and presented in Figures 4.2a-d. The Trajectory Planning (TP) provides an initial guess to solve the previously described optimization problem. As depicted in Figure 5.3b, it is possible to obtain an optimized trajectory.



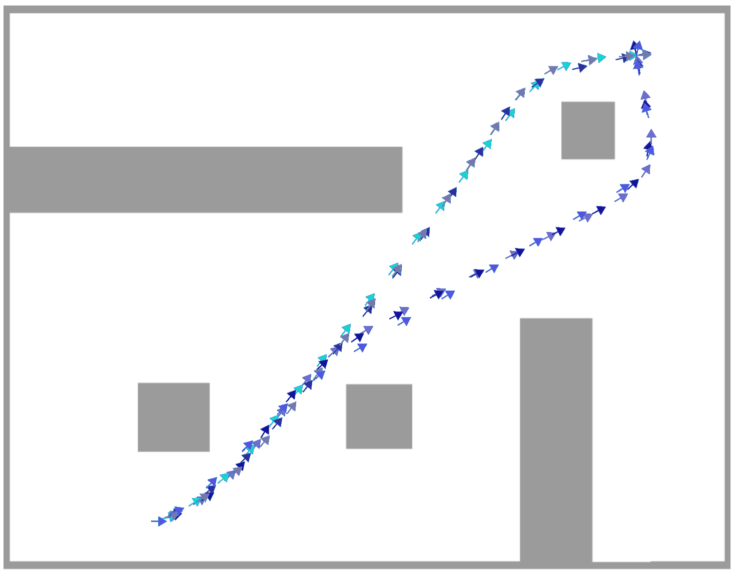
(a) Trajectory Planning



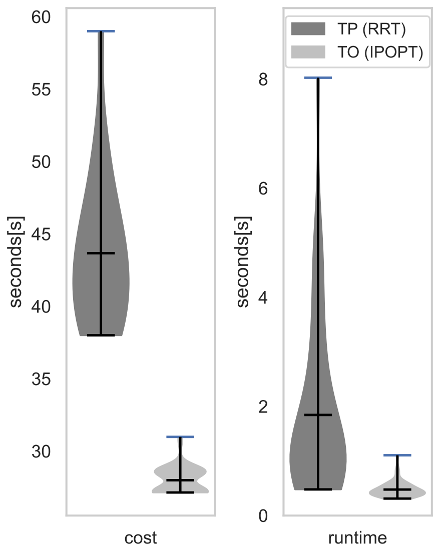
(b) Trajectory Optimization

Figure 5.3: Initial guess trajectory (a) and post optimized trajectory (b).

The motion query presented in Figure 5.3 was solved, over fifty times, using the method presented in this chapter. Figure 5.4a illustrates all the computed trajectories overlaid. It is possible to distinguish between two distinct sets of paths. In reality, the IPOPT algorithm can only generate locally optimal solutions, which are highly dependent on the Kinodynamic RRT algorithm. Figure 5.4b displays the cost, or total trajectory duration, for the TP and Trajectory Optimization (TO) modules. As expected, the cost of the Kinodynamic RRT trajectory is always higher than the optimized one. Additionally, the computational time of the TP and TO modules are also depicted in Figure 5.4b. The TP module is predominantly slower compared to the TO one.



(a) Overlaid optimized trajectories. Two sets of paths can be distnguished. The IPOPT generate locally optimal solutions, which are highly dependent on the Kinodynamic RRT algorithm.



(b) Data distribution regarding trajectory duration and computational time for the Trajectory Planning and Trajectory Optimization modules. The violin plot symbolism used is described in Figure 4.3b.

Figure 5.4: Data obtained with fifty trajectories generated for the same optimization problem. Overlaid fifty optimized trajectories (a) and data distribution regarding trajectory duration and computational time (b) for the Trajectory Planning and Trajectory Optimization modules.

Chapter 6

Multi-Robot Trajectory Optimization

When generating an optimized trajectory in a multi-robot system, it is necessary to ensure that each robot does not collide with any of the static or dynamic obstacles in the environment. In the present chapter, a novel constraint regarding obstacle avoidance is design to address not only static but dynamic obstacles as well. The proposed optimization problem is examined for two, six and fifty-robot problems.

6.1 Optimization Problem Reformulation

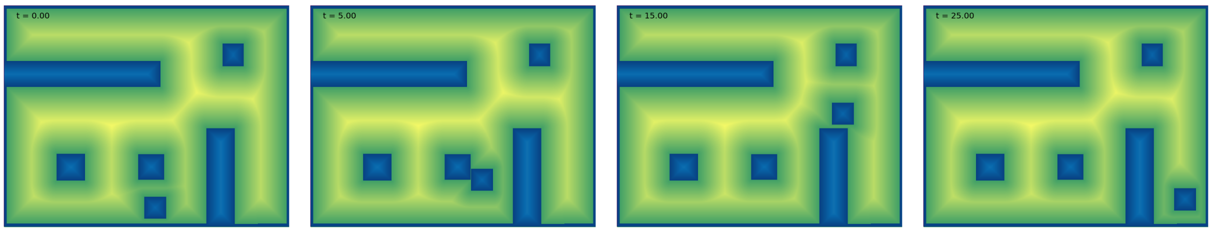
In multi-robot systems it is not only necessary to ensure a safe distance from static obstacles. In this chapter, dynamic obstacle are considered. From now, the dynamic obstacles are expressed as other moving robots. However, any dynamic object, such as people, can also be avoided with this method, if provided with a predictive movement model.

Consequently, for a vehicle R^i it is necessary to ensure that the distance between every position $\mathbf{p}^i(t)$ and all static obstacles and moving robots must respect the safe margin:

$$\begin{aligned} sd(\mathbf{p}^i(t), \mathbf{p}^j(t), O^m) \geq d_{safe}, \quad & \forall i,j \in \{1,...,N\}, \\ & \forall m \in \{1,...,M\}. \end{aligned} \tag{6.1}$$

Let $sdf(t)$ be a function that represents the SDF of the environment at a time instant t, where static obstacles and positions of vehicles with already defined trajectories are considered. To do that, for a specific time instant, a quadrangular shape centred in $\mathbf{p}^i(t)$ is represented as an obstacle in the occupancy gridmap. Figures 6.1a-d illustrate a set of SDFs representing static obstacles and a moving vehicle in the environment at various time instants.

A set of $sdf(t)$ is generated with a defined time interval, $t_q \in \{t_{init}, ..., t_{final}\}$, where $t_{final} = max(t^i_{goal}), \forall i \in$



(a) $sdf(t = 0s)$ (b) $sdf(t = 5s)$ (c) $sdf(t = 15s)$ (d) $sdf(t = 25s)$

Figure 6.1: Set of SDFs illustrating the position of a robot R^i through time. The robot is considered as a moving obstacle in the environment[1].

$\{1, ..., N\}$. Let define a tuple that attaches the time step to the respective generated $sdf(t_q)$:

$$\{(t_q, sdf(t_q))\}, \forall q. \tag{6.2}$$

The set of all generated tuples is given by:

$$\mathcal{SD} = \bigcup_{t_q \in \{t_{init}, \dots, t_{final}\}} \{(t_q, sdf(t_q)\}. \tag{6.3}$$

By interpolating $\mathcal{SD}$, a function that provides as an output the signed distance from R^i to the nearest obstacle or vehicle, given a time t and the respective robot pose $\mathbf{p}^i(t)$, is obtained. This function can be defined as:

$$d(t, \mathbf{p}^i(t)). \tag{6.4}$$

This function differs from the one described in (5.7) due to the fact that the environment now changes over time. Thus, it is crucial that in (6.4) the time instant is given as an input. Thus, it is necessary to revise the optimization problem described in Chapter 5. The only constraint to be modified is the one regarding collision avoidance, more specifically the inequality (5.12). Finally, the multi-robot optimization problem is given by:

$$\begin{aligned}
\text{Minimize} \quad & J(h, \mathbf{x}_k, \mathbf{u}_k) && (6.5)\\
\text{w.r.t.} \quad & h, \mathbf{x}_k, \mathbf{u}_k, \quad \forall k \in \{0, ..., C-1\} \\
\text{subject to} \quad & \mathbf{x}_{k+1} - \mathbf{x}_k = \frac{1}{2} h(f_k + f_{k+1}), \quad \forall k && (6.6)\\
& d(hk, \mathbf{p}(hk)) \geq d_{safe}, \quad \forall k && (6.7)\\
& g(t_{init}, t_{goal}, \mathbf{x}_0, \mathbf{x}_{C-1}) \leq 0 && (6.8)\\
& \mathbf{x}_{min} \leq \mathbf{x}_k \leq \mathbf{x}_{max}, \quad \forall k && (6.9)\\
& \mathbf{u}_{min} \leq \mathbf{u}_k \leq \mathbf{u}_{max}, \quad \forall k && (6.10)
\end{aligned}$$

[1]youtube video URL: https://youtu.be/D9FBw7APRAg

The novelty of this method is that the optimization problem described above presents a single constraint regarding obstacle avoidance, which includes both static and dynamic obstacles. In classic multi-robot trajectory optimization approaches, there is a restriction for each pair of robots that limits the distance between them. In those cases, the computational complexity scales with the number of agents. This novel approach makes a tradeoff between computational effort and allocated memory since it is necessary to precompute and store a set of signed distance fields that describe the dynamic environment. Nevertheless, with this implementation, the number of restrictions does not increase with the number of robots and, consequently, it is possible to reduce the problem's time complexity.

6.1.1 Architecture of Motion Planning and Optimization

Approach A

Typically, the TP with the Kinodynamic RRT presented in Section 4, does not take moving obstacles into account, i.e., when the tree expands, the other agent positions are ignored. When a new node is added to the tree, there is a collision avoidance module that only verifies whether the node collides or not with a static obstacle of the initial map of the environment. Figure 6.2 illustrates the architecture for generating each optimal trajectory of a multi-robot system, in which the inputs are: the updated map of the environment $\mathcal{W}$, the already assigned trajectories of all vehicles $\mathcal{P}$ and the task s^i that is going to be assigned. In this approach, only the TO module considers other vehicles trajectories and, consequently, is the only responsible for the moving obstacle avoidance. The TP takes as inputs the environment static obstacles and the task assigned to vehicle R^i, i.e., the start and goal poses to plan the initial guess trajectory.

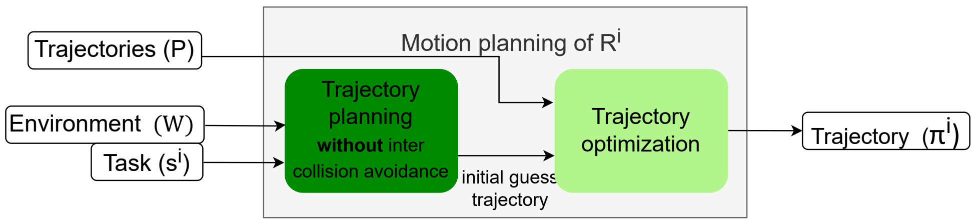


Figure 6.2: Architecture of approach A to generate an optimized trajectory of robot R^i. This framework takes as input the updated map of the environment $\mathcal{W}$, the already defined trajectories of all vehicles $\mathcal{P}$ and the task s^i that is going to be assigned. In this case, the Trajectory Planning only considers static obstacles and does not avoid moving ones. In contrast, the Trajectory Optimization module considers both static and moving obstacles.

Approach B

However, the Kinodynamic RRT collision-avoidance module can be modified to take moving obstacles into account as well. Essentially, before adding a new node to the tree, the distance from that node to the nearest obstacle, static or not, can be determined using $d(t, \mathbf{p}^i(t))$. The time instant and the correspondent cartesian coordinates of the vehicle are defined as t and $\mathbf{p}(t)$, respectively. With this approach, both the TP and TO modules have information regarding other vehicles trajectories. Figure 6.3 represents the architecture of approach B.

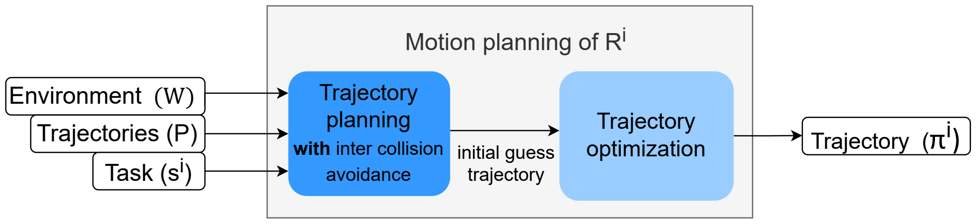


Figure 6.3: Architecture of approach B to generate an optimized trajectory of robot R^i. This framework takes as input the updated map of the environment $\mathcal{W}$, the already defined trajectories of all vehicles $\mathcal{P}$ and the task s^i that is going to be assigned. In this approach, both modules, the Trajectory Planning and Trajectory Optimization, consider static and moving obstacles.

Approach C

As will be analysed in the subsequent sections, there are disadvantages to using only approach A or B. Firstly, approach A's TP is computationally faster. However, often the TO cannot optimize the trajectory given by the TP, since it can be infeasible when considering moving obstacles. Regarding approach B, the altered TP is computationally slower, but leads to fewer fails on the TO. Thus, a third method was design, represented in Figure 6.4, that contemplates both approaches.

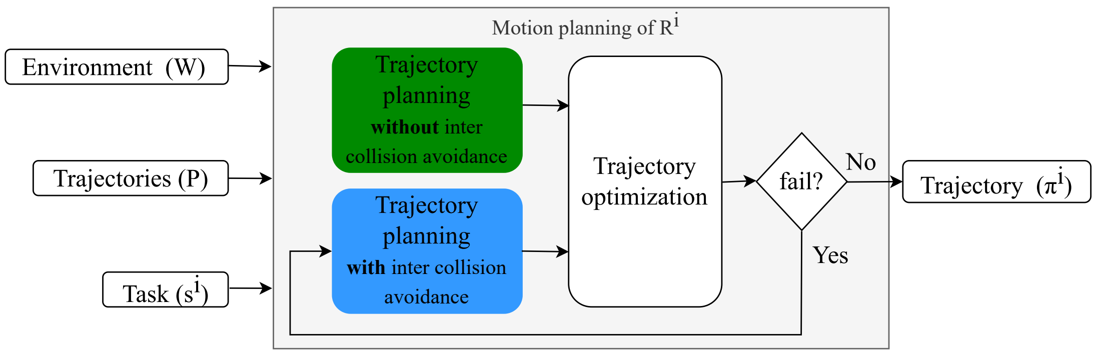


Figure 6.4: Architecture of approach C to generate an optimized trajectory of robot R^i. This framework takes as input the updated map of the environment $\mathcal{W}$, the already defined trajectories of all vehicles $\mathcal{P}$ and the task s^i that is going to be assigned. Initially, the Trajectory Planning only considers static obstacles. If the Trajectory Optimization cannot find a feasible solution, the Trajectory Planning is ran again, considering now both static and moving obstacles.

Firstly, when generating R^i's trajectory, an initial solution using the TP without inter-collision avoidance is calculated. This initial guess is provided to the TO. If the solver cannot achieve an optimal solution, a new initial trajectory is calculated using the TP with inter-collision avoidance and, subsequently, sent to the TO.

6.2 Trajectory Optimization with Two Robots

For a set of two vehicles $\mathcal{R} = \{R^1, R^2\}$, tasks are assigned sequentially to each robot. Firstly, it will be generated π^1 and then π^2. The tasks are assigned at similar time instants, $t^{1,2}_{init} = 0s$. To each vehicle $i = 1, 2$ is assigned an initial pose $\mathbf{x}^i_{init}$ and a goal region $\mathbf{X}^i_{goal}$.

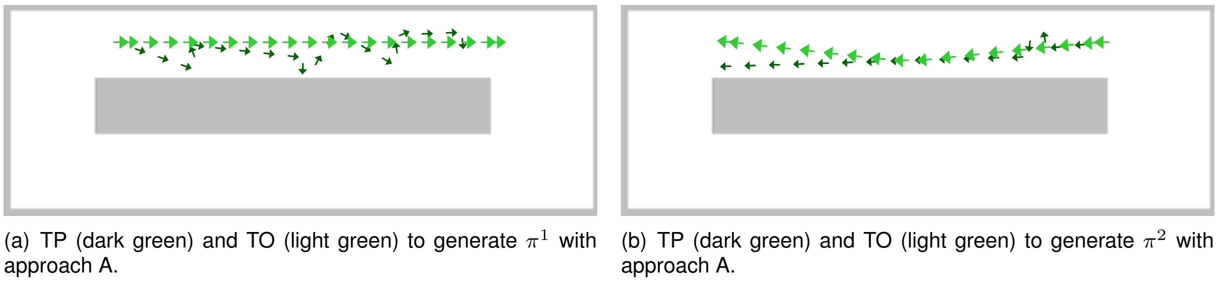
First, a problem is presented in which both vehicles have identical maximum wheel speeds. This issue is solved using approaches A and B. Lastly, two vehicles with distinct maximum wheel speeds are studied.

6.2.1 Two Robots with Identical Maximum Velocities

In this first scenario both vehicles have similar maximum wheels velocities, i.e., $v_{max}^{1,2} = 1ms^{-1}$. Initially, this motion problem is solved with approach A and, posteriorly, with approach B.

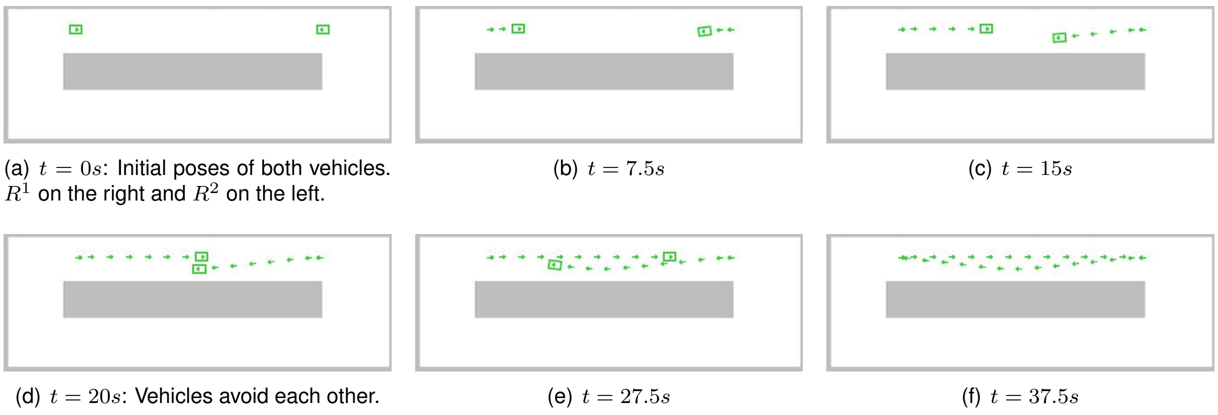
Solution with Approach A

The trajectory of vehicle R^1, illustrated in Figure 6.5a, was the first one to be calculated and, consequently, was treated as a single-agent problem with only static obstacles of the environment to consider. Secondly, Figure 6.5b depicts R^2's trajectory. As formerly explained, with approach A, the TP does not consider moving obstacles when generating the initial guess trajectory (represented in dark green). Nevertheless, the TO takes into account π^1, already defined, and manage to create a trajectory π^2 (represented in light green) that can avoid the previous one. Figures 6.6a-f illustrate the moving agents at different time instants, confirming that they do not collide with each other.


(a) TP (dark green) and TO (light green) to generate π^1 with approach A.

(b) TP (dark green) and TO (light green) to generate π^2 with approach A.

Figure 6.5: Trajectories of robots R^1 and R^2 obtained with approach A.


(a) $t = 0s$: Initial poses of both vehicles. R^1 on the right and R^2 on the left.

(b) $t = 7.5s$

(c) $t = 15s$

(d) $t = 20s$: Vehicles avoid each other.

(e) $t = 27.5s$

(f) $t = 37.5s$

Figure 6.6: Trajectories of R^1 and R^2 at different time instants with approach A. Since there is enough space for both agents to drive through the corridor, the two vehicles swerve from each other.

This motion problem was solved successfully, since both vehicles have enough space to drive through the corridor and pass side by side. Notwithstanding, let us enlarge the safe distance from the agent to the obstacles, so that only one vehicle can fit into the corridor. Figure 6.7a depicts a trajectory π^1 generated as a single agent problem, similar to the previous example. On the other hand, for trajectory

π^2, the TP's first initial guess did not take π^1 into account. As a consequence, the solver was not able to find a feasible local optimal trajectory. In such cases, the IPOPT reaches a maximum number of iterations and returns failure. Figure 6.7b depicts the failed trajectory π^2, while Figures 6.8a-f depict the colliding vehicles' frames.

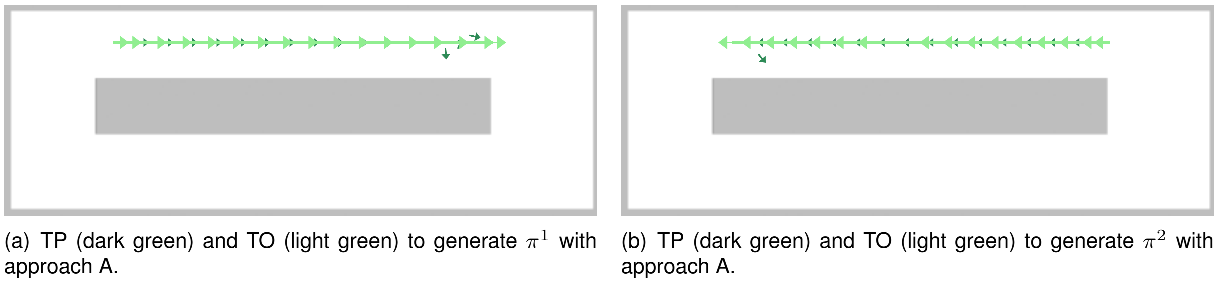

(a) TP (dark green) and TO (light green) to generate π^1 with approach A.
(b) TP (dark green) and TO (light green) to generate π^2 with approach A.

Figure 6.7: Failed trajectories generation after enlarging the safe distance to obstacles with approach A.

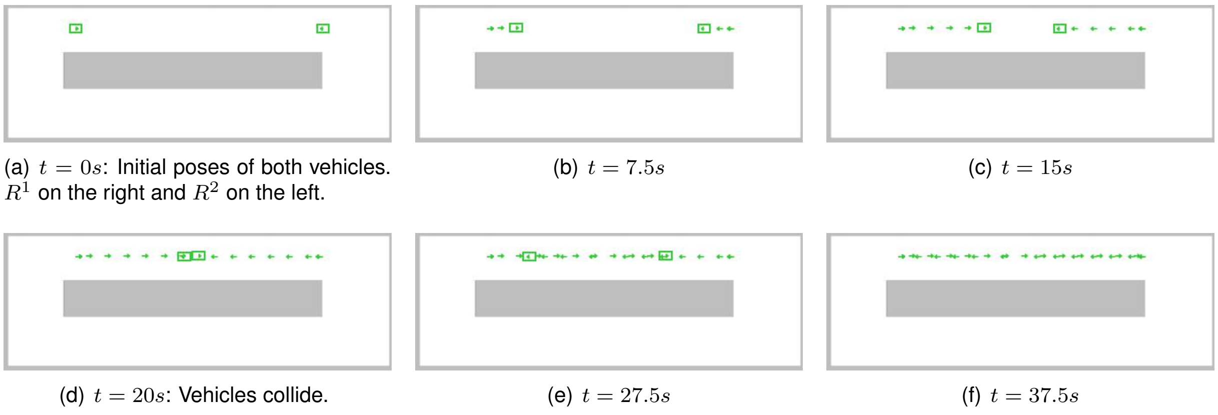

(a) $t = 0s$: Initial poses of both vehicles. R^1 on the right and R^2 on the left.
(b) $t = 7.5s$
(c) $t = 15s$
(d) $t = 20s$: Vehicles collide.
(e) $t = 27.5s$
(f) $t = 37.5s$

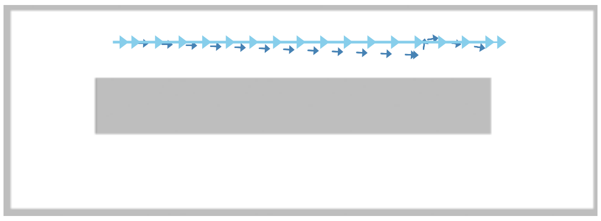
Figure 6.8: Colliding trajectories of robots R^1 and R^2 obtained after enlarging the safe distance to obstacles with approach A. The TO solver could not find a feasible solution for R^2 and returns failure.

Solution with Approach B

Approach B was implemented to prevent the previously described scenario in which the IPOPT is incapable of determining a feasible trajectory. With this architecture, the TP has information regarding static and moving obstacles, such as other vehicles. Figures 6.9a and b show trajectories π^1 and π^2, respectively. In this scenario, the safe distance to the obstacles was also increased. Thus, robot R^2's TP could not create an initial guess trajectory that would have enough space to pass through the corridor. Therefore, a feasible initial trajectory (represented in dark blue) that goes around the central obstacle was obtained. In this case, the solver was able to find a local optimal solution (represented in light blue). Figures 6.10a-c depict the approach B's solution to solve the previously stated problem.

6.2.2 Two Robots with Different Maximum Velocities

Now, consider an environment with only one corridor and no obstacles. In this next scenario, both vehicles have different maximum wheel velocities, i.e., $v^1_{max} = 1ms^{-1}$ and $v^2_{max} = 2ms^{-1}$. The following motion planning problems is solved using approach B.


(a) TP (dark blue) and TO (light blue) to generate π^1 with approach B.

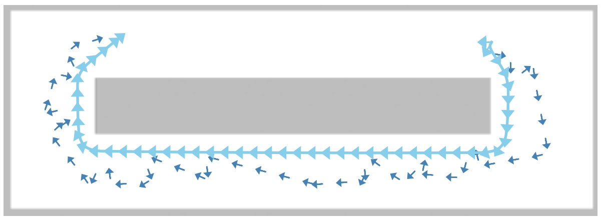
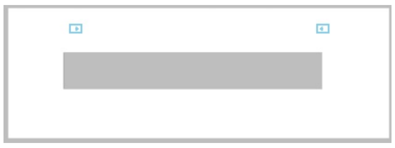
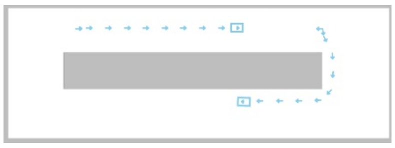

(b) TP (dark blue) and TO (light blue) to generate π^2 with approach B.

Figure 6.9: Trajectories of robots R^1 and R^2 obtained with approach B.


(a) $t = 0s$: Initial poses of both vehicles. R^1 on the right and R^2 on the left.


(b) $t = 25s$

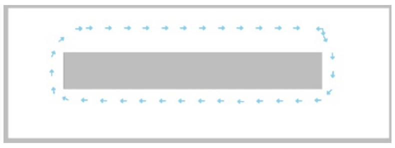
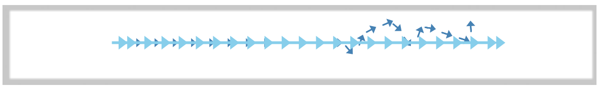

(c) $t = 60s$

Figure 6.10: Trajectories of R^1 and R^2 at different time instants with approach B. Even with not enough space for both agents to pass through the same corridor, both vehicles find feasible routes.

First, assume that the corridor is wide enough for both vehicles to fit side by side. The algorithm was able to find a solution in which the faster vehicle could pass the slower one, as it is verified in Figures 6.11 and 6.12. In Figure 6.13 the velocity of both right and left wheels is presented for every collocation point. As it can be verified, both vehicles try to make use of the maximum velocity of each wheel.


(a) TP (dark blue) and TO (light blue) to generate π^1 with $v^1_{max} = 1ms^{-1}$.

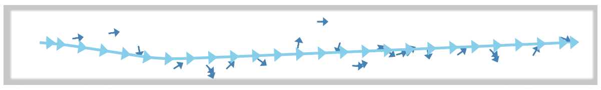

(b) TP (dark blue) and TO (light blue) to generate π^2 with $v^2_{max} = 2ms^{-1}$.

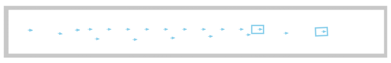
Figure 6.11: Trajectories of robots R^1 and R^2 with different maximum velocities.

(a) $t = 0s$: Initial poses of both vehicles. R^1 on the left and R^2 on the right.

(b) $t = 10s$

(c) $t = 15s$: R^1 passes R^2.

(d) $t = 22.5s$


(e) $t = 30s$

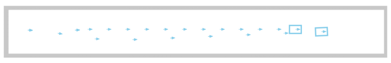
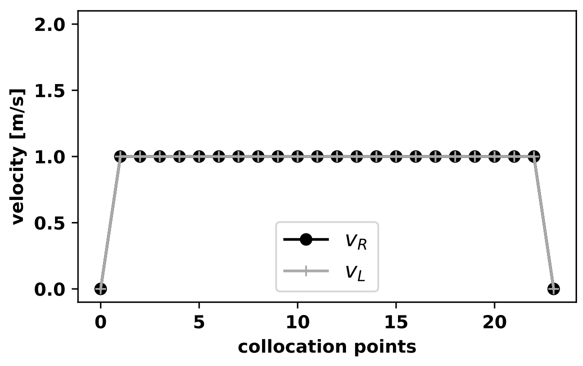
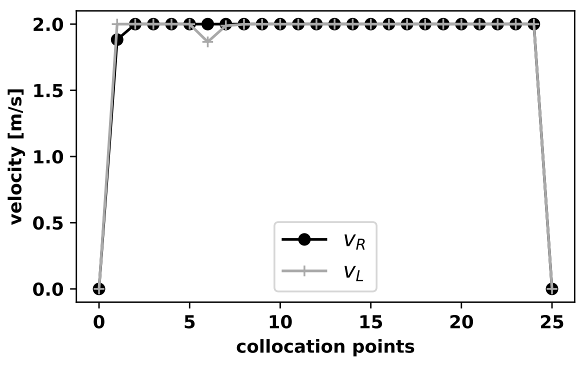

(f) $t = 35s$

Figure 6.12: Trajectories of R^1 and R^2 at different time instants with different maximum velocities, $v^1_{max} = 1ms^{-1}$ and $v^2_{max} = 2ms^{-1}$. The faster vehicle was able to pass the slower one.

Once again, let us increase the safe distance between vehicles so that there is only enough space for one vehicle to drive through the corridor. As shown in Figure 6.14a, the trajectory of vehicle R^1 is the first to be generated. Figure 6.16a proves that R^1 moves with the maximum velocity in both wheels. Next, the trajectory of R^2 is calculated. As shown in Figure 6.16b, R^2 cannot exceed the R^1's velocity while following it. Finally, in Figures 6.15a-c, both vehicles are shown moving in different frames.

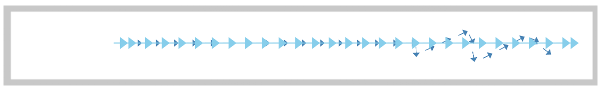


(a) Right and left wheel velocity at every collocation point of vehicle R^1. R^1 reaches the maximum velocity in both wheels.

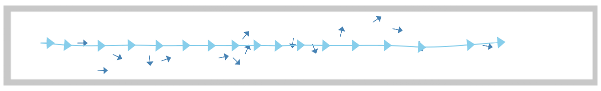


(b) [Right and left wheel velocity at every collocation point of vehicle R^2. R^2 reaches the maximum velocity in both wheels.

Figure 6.13: Right and left wheel velocity at every collocation point of vehicles R^1 and R^2 with different maximum velocities, $v^1_{max} = 1ms^{-1}$ and $v^2_{max} = 2ms^{-1}$.

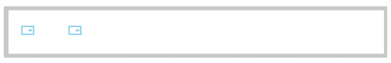


(a) TP (dark blue) and TO (light blue) to generate π^1 with $v^1_{max} = 1ms^{-1}$.

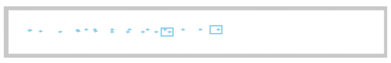


(b) TP (dark blue) and TO (light blue) to generate π^2 with $v^2_{max} = 2ms^{-1}$.

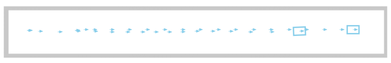
Figure 6.14: Trajectories of robots R^1 and R^2 with different maximum velocities, $v^1_{max} = 1ms^{-1}$ and $v^2_{max} = 2ms^{-1}$.



(a) $t = 0s$: Initial poses of both vehicles. R^1 on the left and R^2 on the right.

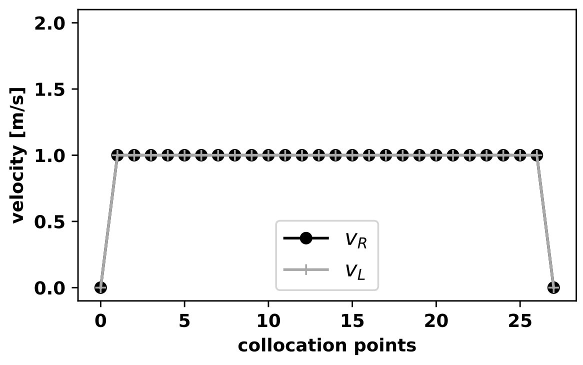


(b) $t = 27.5s$

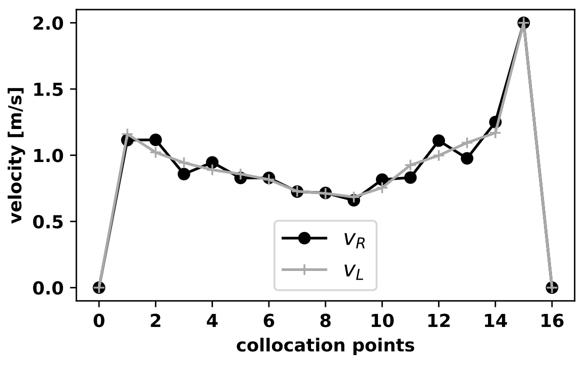


(c) $t = 37.5s$

Figure 6.15: Trajectories of R^1 and R^2 at different time instants with different maximum velocities, $v^1_{max} = 1ms^{-1}$ and $v^2_{max} = 2ms^{-1}$. The faster vehicle was able to follow the slower one.



(a) Right and left wheel velocity at every collocation point of vehicle R^1. R^1 reaches the maximum velocity in both wheels.



(b) Right and left wheel velocity at every collocation point of vehicle R^2. R^2 does not exceed R^1's velocity while following it.

Figure 6.16: Right and left wheel velocity at every collocation point of vehicles R^1 and R^2 with different maximum velocities, $v^1_{max} = 1ms^{-1}$ and $v^2_{max} = 2ms^{-1}$.

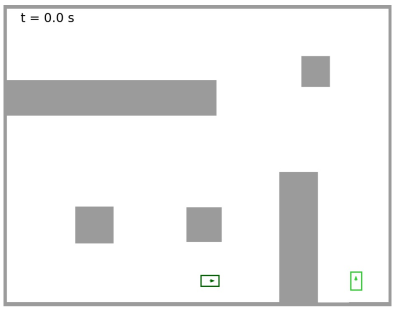
6.3 Trajectory Optimization with Six Robots

For a set of six vehicles $\mathcal{R} = \{R^1, R^2, R^3, R^4, R^5, R^6\}$, asynchronous tasks are assigned at time instants $t^{1,2}_{init} = 0s$, $t^{3,4}_{init} = 5s$ and $t^{5,6}_{init} = 10s$ for the pairs of robots $\{R^1, R^2\}$, $\{R^3, R^4\}$ and $\{R^5, R^6\}$, respectively. The trajectories are generated sequentially, i.e., the first and last trajectories to be calculated are π^1 and π^6, respectively. To each vehicle $i = 1, .., 6$ is assigned an initial pose $\mathbf{x}^i_{init}$ and a goal region $\mathbf{X}^i_{goal}$.

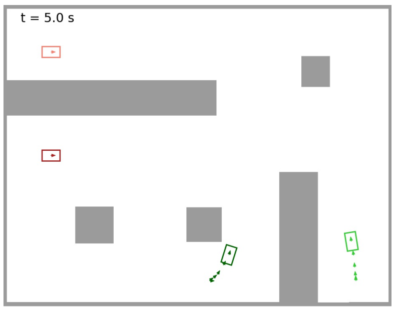
First, the results for the described motion problem are presented without inter-vehicle avoidance. Then, this scenario is solved using the proposed formulation and, therefore, taking into account moving obstacles. Finally, the algorithm scalability is analysed.

6.3.1 Without Inter-Robot Avoidance

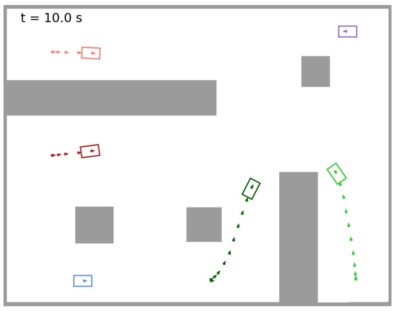
Let one assume that the vehicles do not take into consideration the others trajectories. Therefore, all trajectories are generated as single-agent problems, as described in Chapter 5. Therefore, the optimization problem used is the one described in (5.10)-(5.15), i.e., only static obstacles in the environment are going to be avoided. The results are depicted in Figures 6.17a-f. As expected, there were collisions between vehicles. These results emphasize the importance of implementing a robust mechanism that handles multi-robot systems.



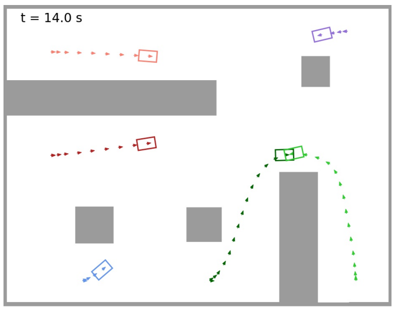
(a) $t = 0s$: R^1 (dark green) and R^2 (light green) begin their trajectories.



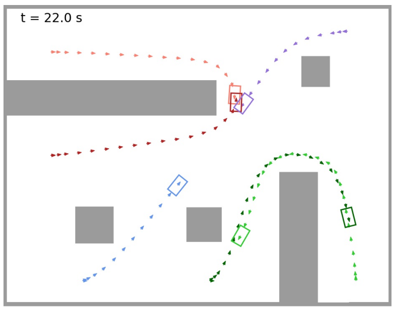
(b) $t = 5s$: R^3 (orange) and R^4 (red) begin their trajectories.



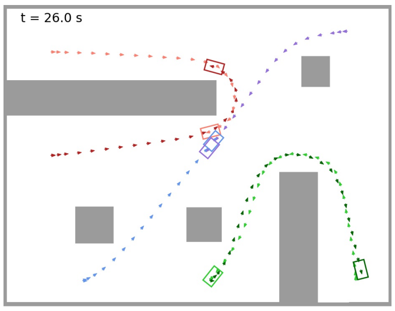
(c) $t = 10s$: R^5 (blue) and R^6 (purple) begin their trajectories.



(d) $t = 14s$: R^1 (dark green) and R^2 (light green) collide with each other.



(e) $t = 22s$: R^3 (orande), R^4 (red) and R^5 (blue) collide with each other.



(f) $t = 26s$: R^5 and R^6 collide with each other.

Figure 6.17: Trajectories of six vehicles without inter-vehicle avoidance, at different time instants. The robots cannot swerve to avoid collisions.

6.3.2 With Inter-Robot Avoidance

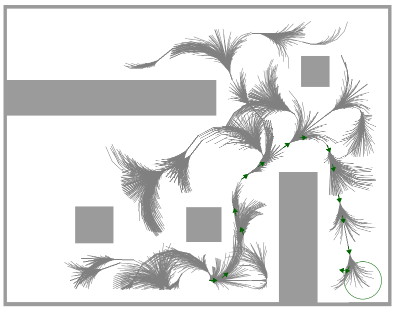
Since the trajectory of R^1 is the first one to be calculated, it does not depend on any other vehicle. Therefore, it is treated as a single-agent problem. The tree and initial trajectory generated by the Kynodinamic RRT are illustrated in Figure 6.18a. Figure 6.18b shows the final optimized trajectory.

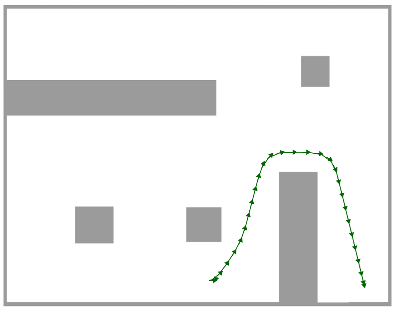
When generating trajectory π^2, the position of R^1 through time is described by a set of SDFs. Figures 6.1a-d depicts some of these SDFs. This set is interpolated and used in the optimization problem to reveal information regarding the distance from R^2 to the nearest obstacle or vehicle. Figures 6.18c and 6.18d illustrate the initial guess trajectory and the optimized π^2, respectively. Similarly to π^2, R^3's trajectory is generated by avoiding the already existing ones, π^1 and π^2. The remaining trajectories were generated using the same reasoning as the preceding ones. Figures 6.18e-i depict the initial guess and final locally optimal trajectories of vehicles R^3, R^4, R^5, and R^6. Finally, Figures 6.19a-f show some strategic frames that validate the implementation. It is possible to verify that all vehicles swerve to avoid collisions.

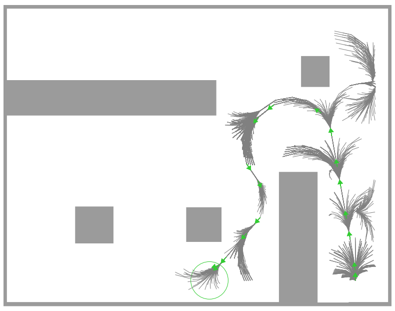
To gather information regarding the implemented method, each trajectory of every vehicle was generated fifty times. The motion planning problem of each robot was kept unchanged for every sample. The overlapping trajectories of each vehicle are depicted in Figures 6.20a-f.

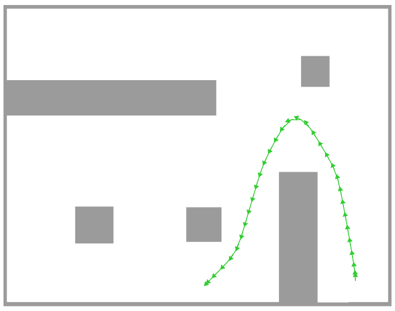
The six-vehicle motion planning problem of set $\mathcal{R}$ was solved fifty times with both approaches A and B in order to compare them. The results are illustrated in Figures 6.21a-c. The data regarding the computational time (Figure 6.21a) and the trajectory duration (Figure 6.21b) of each module (TP and TO), for both approaches, was gathered for each of the six agents. Finally, the frequency of failed trajectory generations is plotted in Figure 6.21c. A failure occurs when the IPOPT solver of the TO module cannot find a locally optimal trajectory given the initial guess provided by TP.

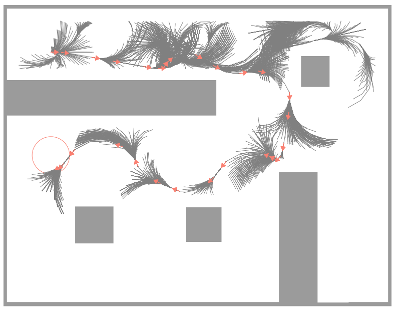
Firstly, concerning approach A, it can be verified that the TP module is computationally faster. However, the TO of approach A cannot always optimize the trajectory given by the TP, since it can be infeasible when considering moving obstacles. This leads to a higher frequency of failures. Regarding approach B, the altered TP is computationally slower, but the solver can always find a solution. Thus, it can be concluded that there are both advantages and disadvantages when using either approach A or B. Thus, approach C was design, represented in Figure 6.4, that contemplates both approaches. The following tests depicted in this dissertation will be solved using the architecture corresponding to approach C.

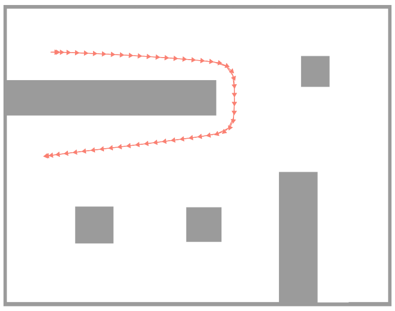

(a) Trajectory planning of R^1. Kinodynamic RRT tree constructed (grey) and generated initial guess trajectory (dark green).

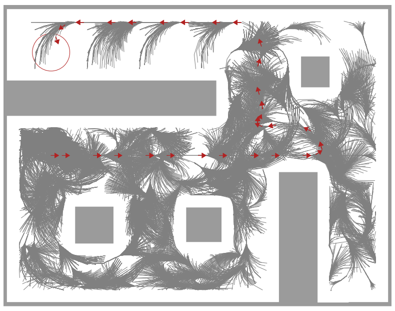

(b) Trajectory optimization of R^1. Trajectory π^1 is the first to be calculated, thus it does not depend on any other vehicle.

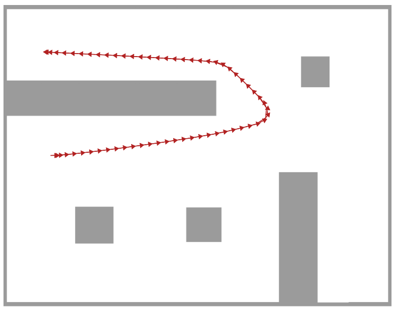

(c) Trajectory planning of R^2. Kinodynamic RRT tree constructed (grey) and generated initial guess trajectory (light green).

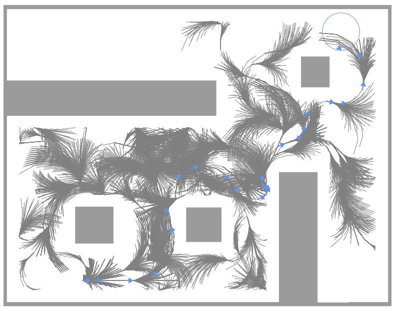

(d) Trajectory optimization of R^2. Trajectory π^2 is generated by avoiding the already existing one, π^1.

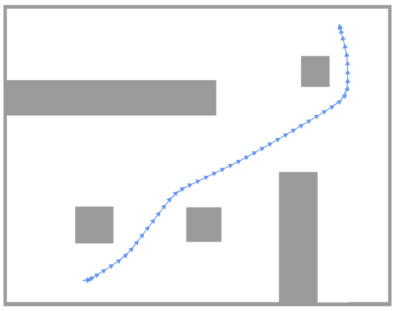

(e) Trajectory planning of R^3. Kinodynamic RRT tree constructed (grey) and generated initial guess trajectory (orange).

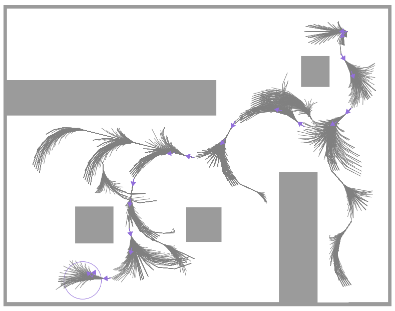

(f) Trajectory optimization of R^3. Trajectory π^3 is generated by avoiding the already existing ones, π^1 and π^2.


(g) Trajectory planning of R^4. Kinodynamic RRT tree constructed (grey) and generated initial guess trajectory (red).


(h) Trajectory optimization of R^4. Trajectory π^4 is generated by avoiding the already existing ones, π^1, π^2 and π^3.


(i) Trajectory planning of R^5. Kinodynamic RRT tree constructed (grey) and generated initial guess trajectory (blue).


(j) Trajectory optimization of R^5. Trajectory π^5 is generated by avoiding the already existing ones, π^1, π^2, π^3 and π^4.


(k) Trajectory planning of R^6. Kinodynamic RRT tree constructed (grey) and generated initial guess trajectory (purple).

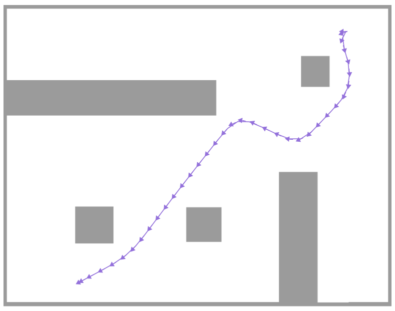
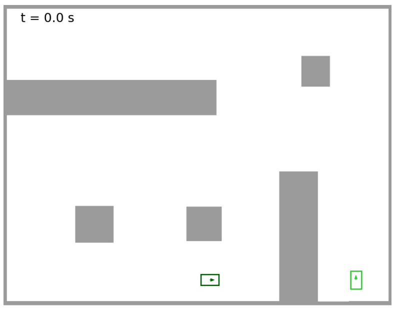
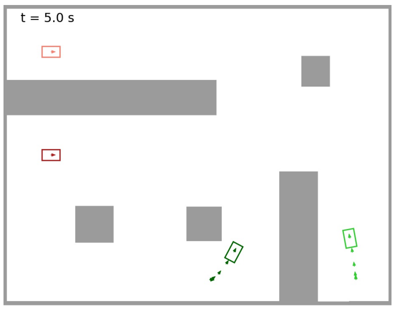

(l) Trajectory optimization of R^6. Trajectory π^6 is generated by avoiding the already existing ones, π^1, π^2, π^3, π^4 and π^5.

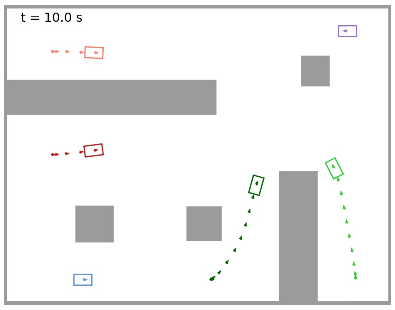
Figure 6.18: Generation of trajectories π^1, π^2, π^3, π^4, π^5 and π^6.



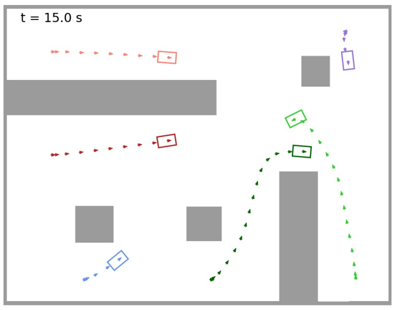
(a) $t = 0s$: R^1 (dark green) and R^2 (light green) begin their trajectories.



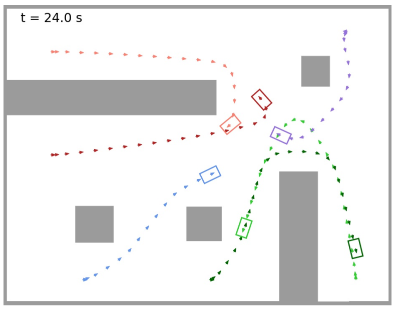
(b) $t = 5s$: R^3 (orange) and R^4 (red) begin their trajectories.



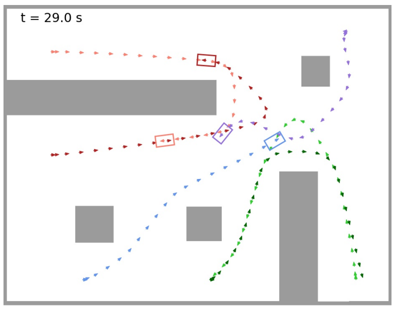
(c) $t = 10s$: R^5 (blue) and R^6 (purple) begin their trajectories.



(d) $t = 15s$: R^1 (dark green) and R^2 (light green) avoid colliding with each other.

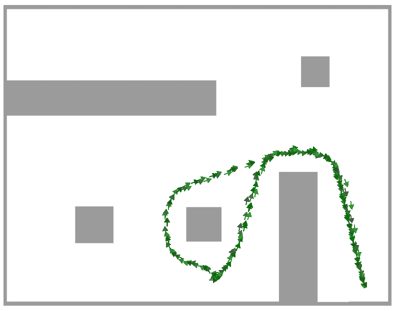


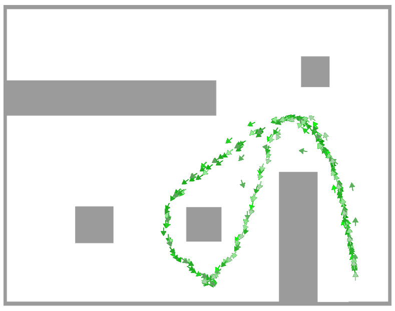
(e) $t = 24s$: R^3 (orange), R^4 (red) avoid colliding with each other.

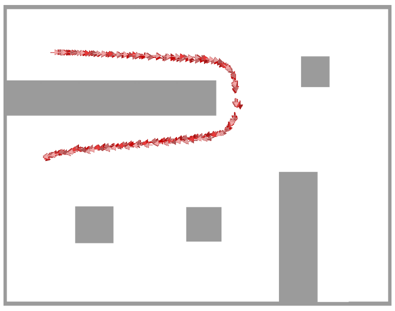


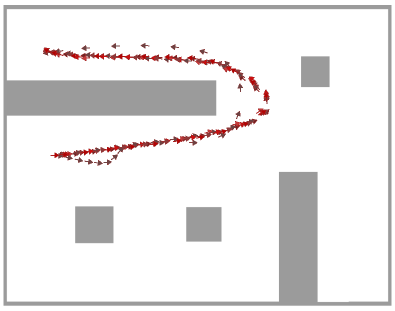
(f) $t = 29s$: R^5 (blue) and R^6 (purple) avoid colliding with each other.

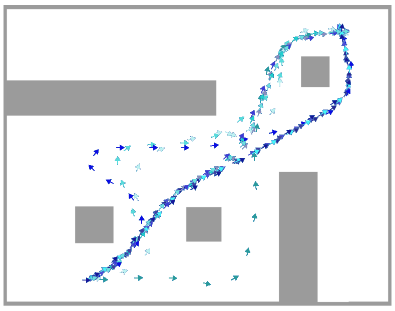
Figure 6.19: Trajectories of six vehicles with inter-vehicle avoidance, at different time instants. All robots swerve to avoid collisions [2].


(a) Overlaid π^1


(b) Overlaid π^2


(c) Overlaind π^3


(d) Overlaid π^4


(e) Overlaind π^5

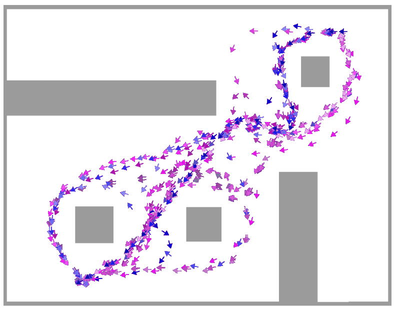
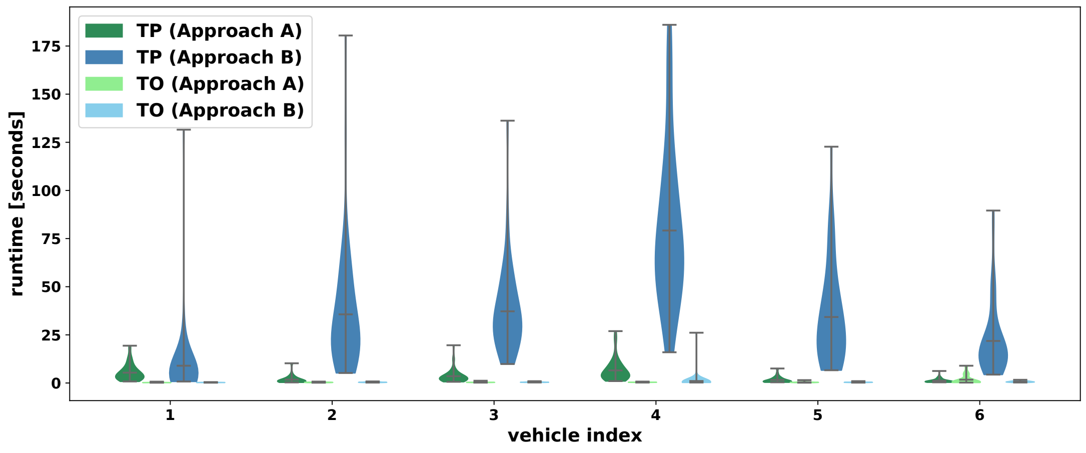
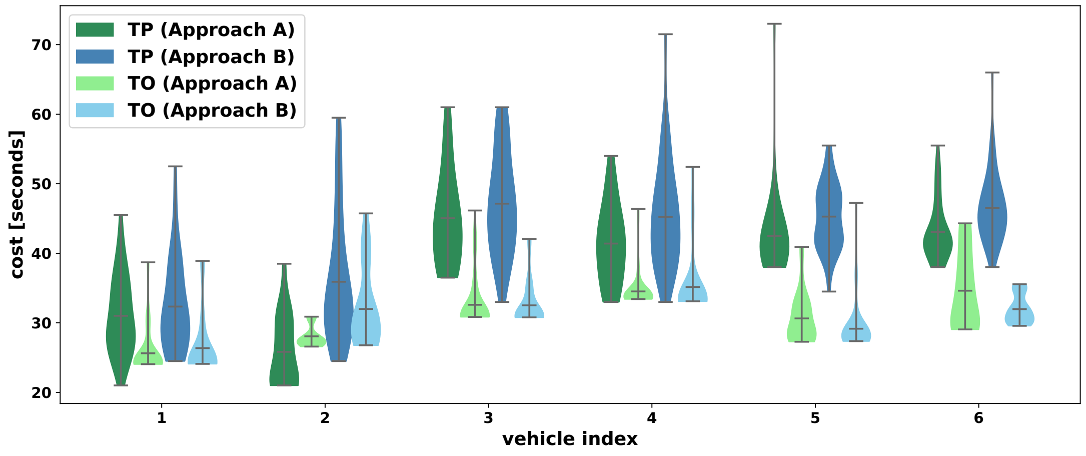

(f) Overlaid π^6

Figure 6.20: Fifty optimized trajectories overlapped for each motion planning problem with inter-vehicle avoidance.

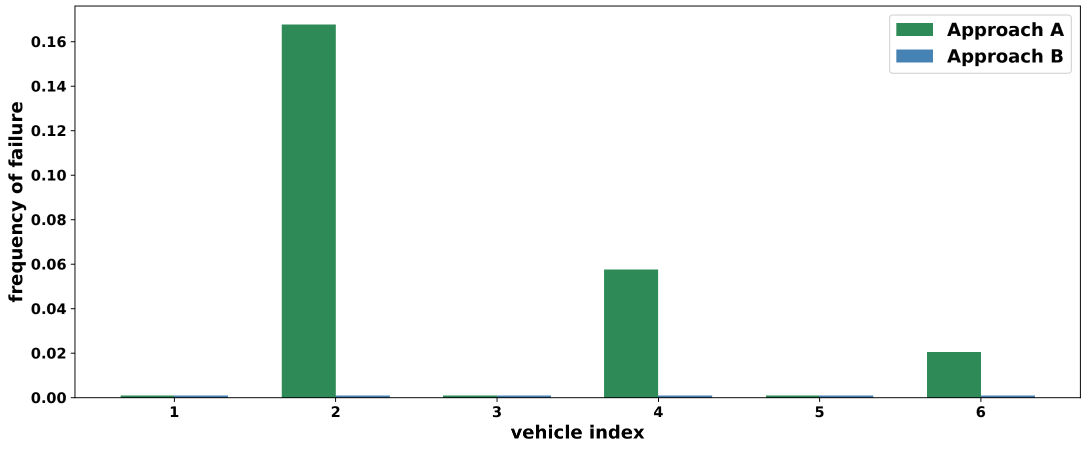
[2]youtube video URL: https://youtu.be/-77nMGYzww8



(a) Data distribution regarding computational time. The violin plot symbolism used is described in Figure 4.3b.



(b) Data distribution regarding trajectory duration (cost). The violin plot symbolism used is described in Figure 4.3b.



(c) Frequency of failure with approaches A and B.

Figure 6.21: Violin plots regarding computational time (a) and trajectory duration (b) for the TP and TO algorithms of each vehicle. Frequency of failure for both approaches (c).

6.4 Scalability Study

Next, the algorithm's behaviour to overcrowded spaces and how many routes can be built on this specific map is examined. The pseudocode that calculate the maximum number of generated trajectories is accounted in Algorithm 8. Firstly, the function $GenerateRandomInitialGoalStates$ computes randomly the initial and final states of each trajectory (line 3). Nevertheless, some conditions must be met, such as:

- Both states must be inside the set of free state space, $\mathbf{x}^i_{init}, \mathbf{x}^i_{goal} \in \mathcal{X}_{free}$;
- All initial poses are separated by a distance greater than a safe margin, $|\mathbf{p}^i_{init} - \mathbf{p}^j_{init}| > d_{safe}$, $\forall i, j \in \{1, ..., N\}$;
- All trajectories are assigned at the same instant, $t^{1,...,N} = 0s$.

For each vehicle R^i, the motion planning module is executed based on approach C (line 4). If a trajectory can be successfully generated, then the process is repeated for a new agent, R^{i+1} (line 8). If trajectory π^i cannot be found, the number of fails is increased (line 6), until a maximum value, n_{max}, is reached. In this case, the algorithm stops the search since it cannot add any more agents to the environment.

Algorithm 8 Maximum number of generated trajectories

```
    Input: state space $\mathcal{X}$
    Ouput: number of generated trajectories $i$

 1: $i = 0, \quad n_{fail} = 0$
 2: while $n_{fail} < n_{max}$ do
 3:     $\mathbf{x}^i_{init}, \mathbf{x}^i_{goal} \leftarrow$ GenerateRandomInitialGoalStates($\mathcal{X}_{free}$)
 4:     $\pi^i \leftarrow$ MotionPlanning($\mathbf{x}^i_{init}, \mathbf{x}^i_{goal}$)
 5:     if $\pi^i$ is failure then
 6:         $n_{fail} \leftarrow n_{fail} + 1$
 7:     end if
 8:     $i \leftarrow i + 1$
 9: end while
10: return $i$
```

This procedure that analyses the scalability of the method was repeated 20 times. The histogram presenting the frequency of the maximum number of agents allowed in the environment can be found in Figure 6.22. The case with the least number of agents was obtained with 16 vehicles. On the other hand, the situation in which more agents were added to the scenario was reached with 25 vehicles. Examples of 16, 20 and 25 agents are represented in Figures 6.23a-c, respectively.

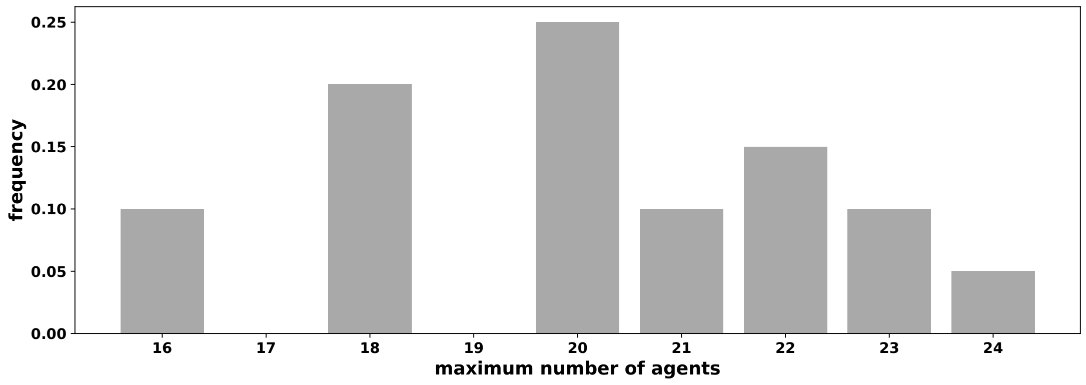


Figure 6.22: Histogram with frequency of maximum number of agents that can be added to the studied scenario.

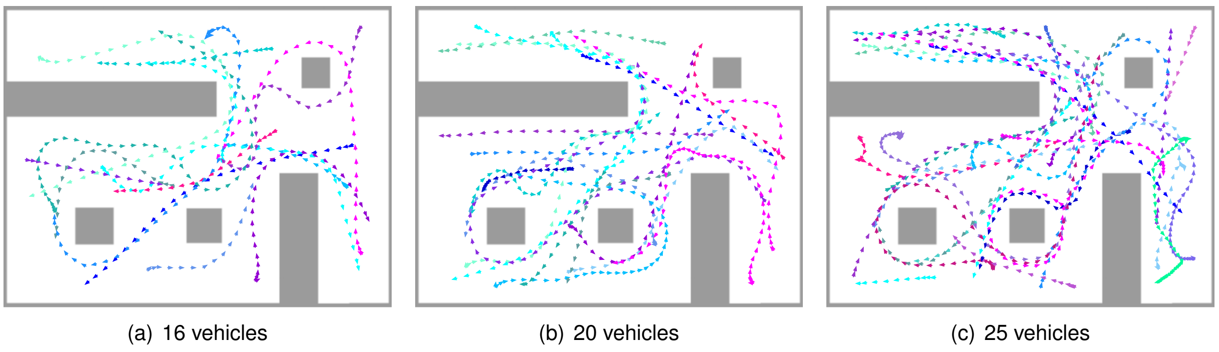


Figure 6.23: Example of scenarios for different maximum number of agents allowed in the environment.

6.5 Trajectory Optimization with Fifty Robots

A set of fifty vehicles, $\mathcal{R} = \{R^1, ..., R^{50}\}$, was used to analyse the computational complexity of the TO module with an increasing number of agents. The initial and final states of the fifty vehicles were generated randomly, respecting function $GenerateRandomInitialGoalStates$'s conditions presented above. One additional condition was considered: ensuring that the distance between initial and final poses of each robot was inside a defined range, $|\mathbf{p}^i_{init} - \mathbf{p}^i_{goal}| < d_{range}$. This allowed to create trajectories of similar lengths.

Unlike the previous case, in which the behaviour of the proposed method in an overcrowded environment was studied, now the purpose is to analyse the computational complexity with an increasing number of agents. Therefore, the map of the environment was enlarged and the boundaries were removed. Figure 6.24 represents the enlarged map, as well as a solution for the fifty-agent motion planning problem.

In addiction, the fifty-agent motion planning problem was solved fifty times for each agent. The data regarding the TO computational time of each agent was collected and presented in Figure 6.25a. As previously mentioned, the first trajectory generated is treated as a single-robot problem. This leads to visibly faster computation compared to the remaining agents. Generally, the computational time of the TO increases slightly with the addition of new vehicles. In fact, the environment is becoming progressively more crowded and the space for vehicles to move more scarce. Figure 6.25b depicts the data

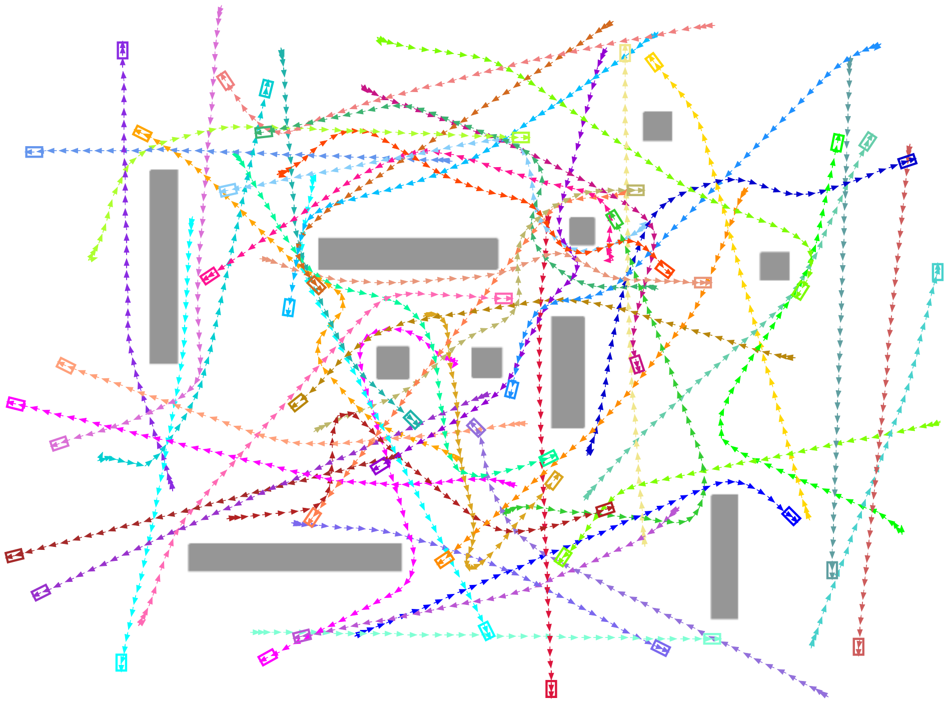
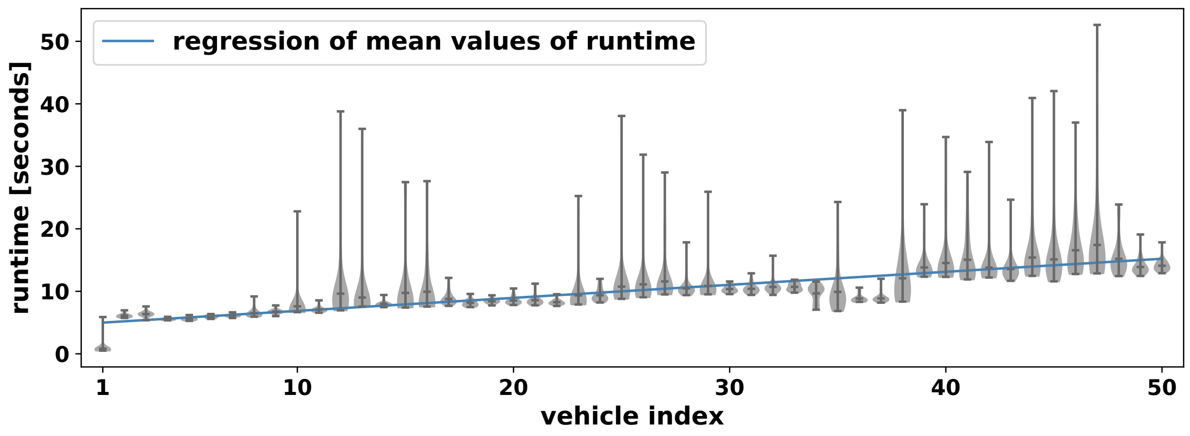


Figure 6.24: Solution for the fifty-multi agent problem. The trajectories of all vehicles were overlapped. The assigned initial pose of each robot is represented by its respective rectangular shape [3].

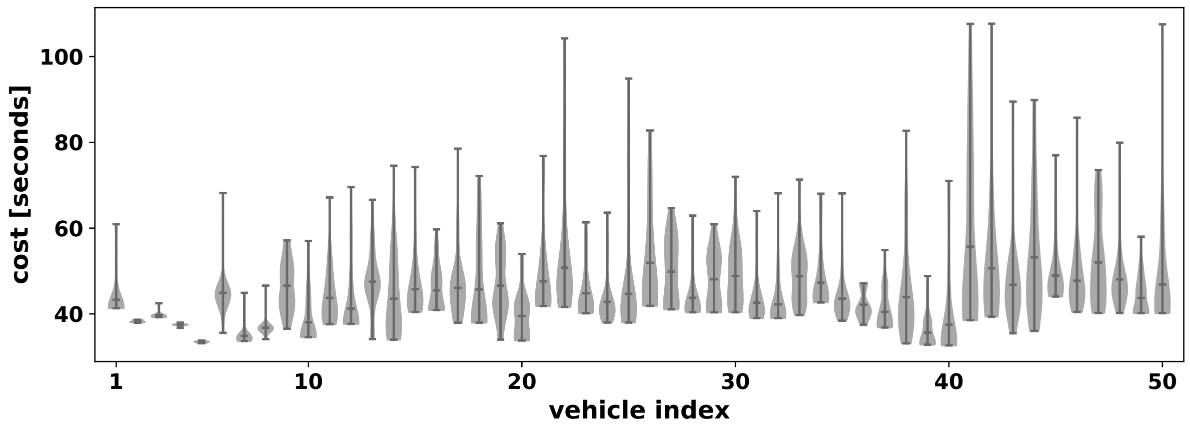
collected regarding the trajectory duration of the generated routes for each vehicle.

Furthermore, a regression analysis of the mean values of computational time of each vehicle was conducted to measure the relationship between the complexity and the number of agents. The resulting regression line, presented in Figure 6.25a, enabled us to estimate a growth rate of 0.208 seconds per agent. This result indicates that the complexity scales slower than an unitary linear rate with the number of vehicles.

[3]youtube video URL: https://youtu.be/b6iUmlzqf0Q



(a) Data distribution regarding computational time of the Trajectory Optimization module for every vehicle. The violin plot symbolism used is described in Figure 4.3b. The regression analysis of the mean values of computational time of each vehicle is represented in blue. The resulting regression line presents a growth rate of 0.208 seconds per agent.



(b) Data distribution regarding trajectory duration obtained for every vehicle. The violin plot simbolism used is described in Figure 4.3b.

Figure 6.25: Data collected regarding the Trajectory Optimization of each vehicle, generated fifty times.

Chapter 7

Real Environment Requirements

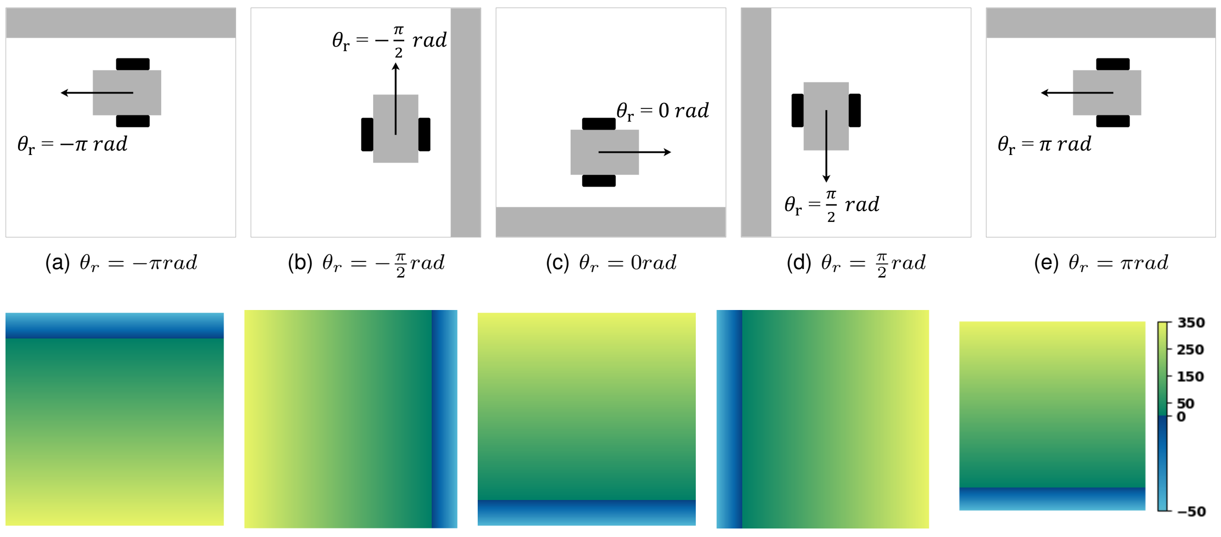
The effectiveness of the proposed method has been validated in simulation. However, when implementing this algorithm in a real-world industrial scenario, some modifications and reformulations are required. This chapter discusses the adjustments made. In Section 7.1, the objective function is reformulated so that robots drive, preferentially, on the right side of the corridors. Section 7.2 presents a distributed dynamic map update algorithm so that all robots can share detected changes in the environment.

7.1 Objective Function Reformulation

In a real-world industrial scenario, it is necessary to ensure the transportation of materials inside the factory. This is done by generating smooth trajectories through the factory's corridors that can avoid static obstacles and other vehicles. Nevertheless, the AMRs are required to drive on the right side of the factory's corridors. This condition contributes to the organization and road-like circulation of the AMRs in the industrial facilities. Additionally, this leads to a more predictable environment for people or other vehicles that might move through the corridors, besides AMRs. Although it is necessary to guarantee that each AMR drives preferably on the right side of the corridor, this must not be a strict restriction since the vehicle should be able to swerve obstacles that might force the robot to momentarily left-hand drive.

Therefore, some adjustments in the cost function must be made. A term was added to the objective function, which consists of a penalty parameter that penalizes when the vehicle is not moving on the right side. First, the measure of violation was conceptualized for a set of vehicle's orientations, such as $\Theta = \{-\pi, -\pi/2, 0, \pi/2, \pi\} rad$. Figures 7.1a-e depict the vehicle in various poses with Θ orientations. Additionally, it is also represented where virtual obstacle/wall/corridor limitation should be placed to ensure that the vehicle drives on its right-hand. Furthermore, the SDF of each obstacle in Figures 7.1a-e was generated and it is depicted in Figures 7.1f-j, respectively. Let $osdf(\theta_r)$ be a function that represents the SDF generated by the right-side obstacle when the vehicle has an orientation θ_r.

For example, when a vehicle is moving with $\theta_r = 0 rad$, as shown in Figure 7.1c, the respective SDF



(a) $\theta_r = -\pi rad$ (b) $\theta_r = -\frac{\pi}{2} rad$ (c) $\theta_r = 0 rad$ (d) $\theta_r = \frac{\pi}{2} rad$ (e) $\theta_r = \pi rad$

(f) $osdf(\theta_r = -\pi rad)$ (g) $osdf(\theta_r = -\frac{\pi}{2} rad)$ (h) $osdf(\theta_r = 0 rad)$ (i) $osdf(\theta_r = \frac{\pi}{2} rad)$ (j) $osdf(\theta_r = \pi rad)$

Figure 7.1: Representation of where the closest obstacle should be placed to ensure that the vehicle drives on its right-hand for the set of vehicle's orientation $\Theta = \{-\pi, -\pi/2, 0, \pi/2, \pi\} rad$, (a)-(e). SDFs generated by the right side obstacle when the vehicle has an orientation from set Θ, (f)-(j).

generated by the right-side obstacle is represented in Figure 7.1h. It can be verified that if the vehicle is moving far away from the right side (yellow color), the value given by $osdf(\theta_r = 0rad)$ is much larger compared to a vehicle that is moving next to the obstacle (green color). This value can be used as a measure of violation to be applied to the cost function. This parameter increases with the distance from the right side of the corridor and is closest to zero in the region where the vehicle is moving at exactly the safe distance from its right-side obstacle or corridor limits.

The set of all generated SDFs presented in Figures 7.1f-j is given by:

$$\mathcal{OSD} = \bigcup_{\theta \in \Theta} \{osdf(\theta)\}. \tag{7.1}$$

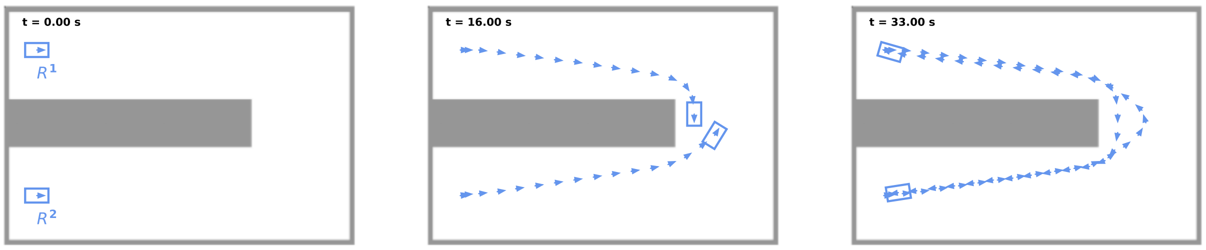
By interpolating $\mathcal{OSD}$, a function that provides the measure of violation is obtained:

$$m(\mathbf{p}_k), \tag{7.2}$$

where the robot pose, $\mathbf{p}_k = (x_k, y_k, \theta_k)$ at a collocation point $k \in \{0, ..., C-1\}$, is received as inputs. The violation term provided by this function is considered at every collocation point. The reformulated cost function is given by:

$$J(h, \mathbf{x}_k, \mathbf{u}_k) = h \sum_{k=0}^{C-1} m(\mathbf{p}_k), \quad \forall k \in \{0, ..., C-1\}. \tag{7.3}$$

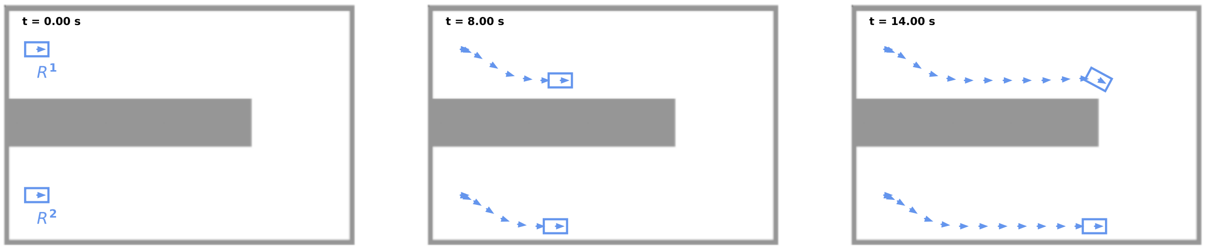
Tasks are assigned sequentially to each robot in a set of two vehicles, $\mathcal{R} = \{R^1, R^2\}$. In Figures 7.2 is presented the solution using the objective function proposed in Chapter 6 that minimizes the trajectory's duration.



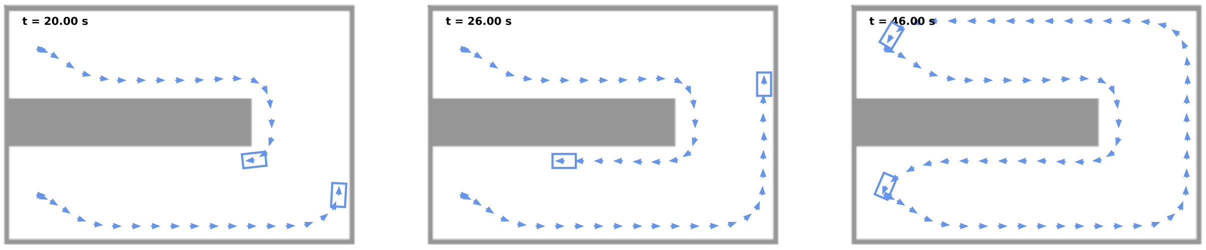
(a) $t = 0s$ (b) $t = 16s$ (c) $t = 33s$

Figure 7.2: Solution using the method proposed in Chapter 6, in which the objective function only minimizes the trajectory duration.

Additionally, for the same motion planning problem, a solution was obtained using the reformulated objective function, in which the vehicles drive preferably on the right side of the corridors. This result is illustrated in Figures 7.3a-f.



(a) $t = 0s$ (b) $t = 8s$ (c) $t = 14s$



(d) $t = 20s$ (e) $t = 26s$ (f) $t = 46s$

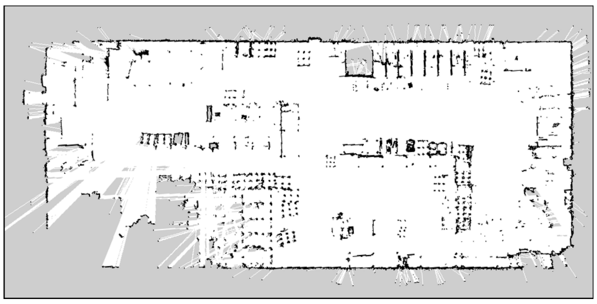
Figure 7.3: Solution using the reformulated method, in which the vehicles drive preferably on the right side on the corridors.

7.2 Dynamic Map Update

Since the environment under study is subject to changes, it is crucial that an autonomous agent perceives and acts accordingly in its environment. To have an accurate picture of the scenario, these changes must be found and then added to the map that is shared by all agents.

Initially, all vehicles own a Static Map (SM). In Figure 7.4a is presented the original map of Imeguisa's installations. The SM is created by binerazing the environment into unoccupied and obstacle areas. The respective SM of Imeguisa's installations, with the available corridors, is presented in 7.4b.

When driving through the environment each AMR computes an occupancy gridmap using laser scanners


(a) Map of Imeguisa's infrastructures.

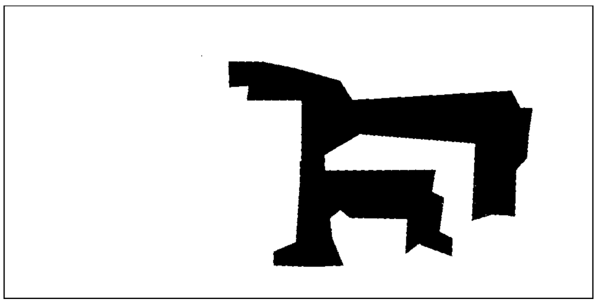
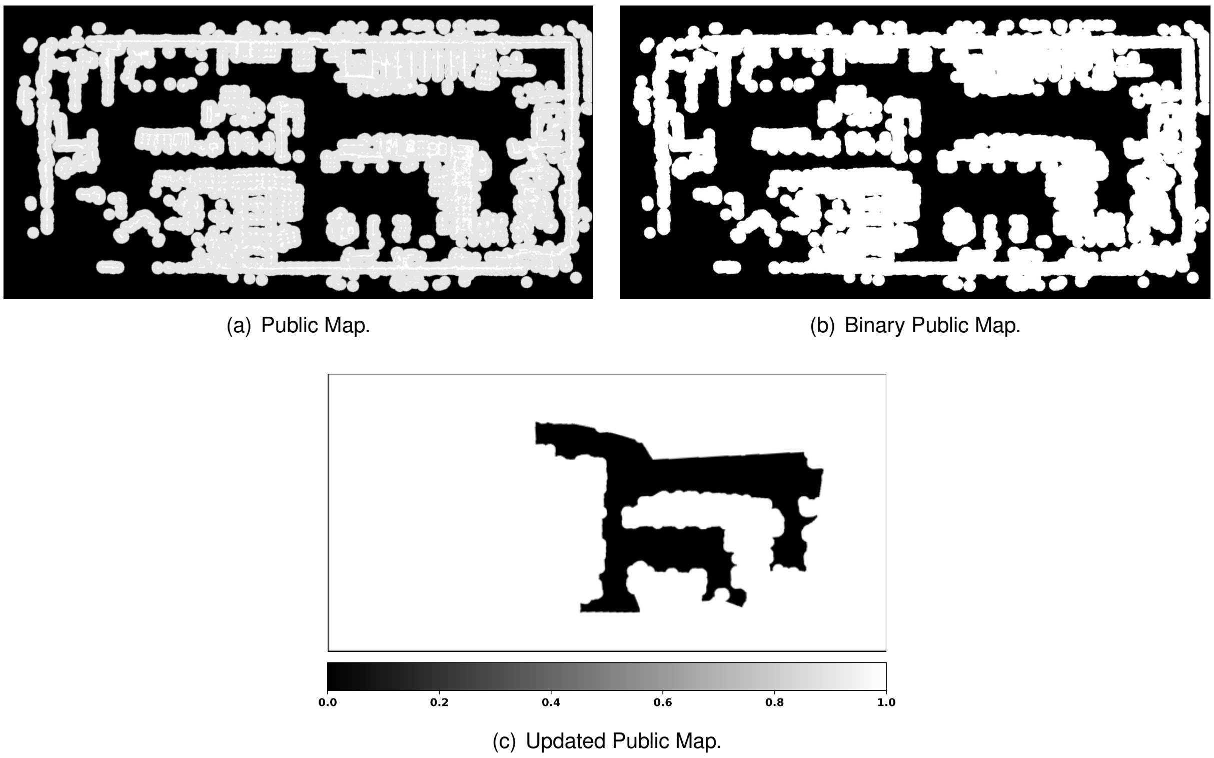

(b) Static map.

Figure 7.4: Original map of Imeguisa's installations (a) and corresponding binary Static Map created (b), in which white represents the obstacle areas and black the unoccupied areas.

Algorithm 9 Decentralized Dynamic Map Update

```
   Input: $SM$, $LM^i$ for all $i = 1, ..., N$
   Output: $UPM$
1: $PM \leftarrow (LM^1 + ...LM^N)/N$
2: $BPM \leftarrow$ Convert $PM$ to a binary map using a threshold $\beta$
3: $UPM \leftarrow SM \vee BPM$
4: return $UPM$
```

built in the mobile robot. A costmap presents the probability of each map cell containing an obstacle. This map is denominated as Local Map (LM) and every robot R^i, $i = \{1, ..., N\}$ creates its own LM^i when driving through the environment. Each LM^i is sent over to the network and shared with others. The decentralized dynamic map update algorithm, presented in Algorithm 9, constructs an updated map of the environment. Each vehicle receives the LM^i of all robots. Then, a Public Map (PM) is created by calculating an average of all LM^i (line 1). An example of a PM is presented in Figure 7.5a. A threshold was defined in order to transform PM into a Binary Public Map (BPM) (line 2). If the value of a PM cell is smaller than this threshold, then it is assigned the value 0 to that cell that corresponds to an unoccupied area. On the other hand, if the value of a PM cell is higher than the threshold, it is assumed that there is an obstacle in that location and the value of the cell is assigned to 1. An example of a BPM is presented in Figure 7.5b. Finally, the SM and the BPM are merged using the logical OR operator (line 3), resulting in the Updated Public Map (UPM). With the UPM all robots can plan their trajectories with the shared information.



(a) Public Map.

(b) Binary Public Map.

(c) Updated Public Map.

Figure 7.5: Public Map (a) computed by the average of all robot's Local Maps and corresponding Binary Public Map (b). The Updated Public Map (c) is generated by merging the Static Map with the Binary Public Map.

Chapter 8

Real Environment Results

In this chapter, the results of the experimental tests in a real environment were conducted at Imeguisa's installations, with a maximum of three vehicles, are depicted. Three different scenarios are presented for a maximum of three AMRs and three tasks to be allocated.

8.1 Experimental Setup

The experimental tests on the work accomplished were conducted at Imeguisa's installations. The map of the environment was already presented in Chapter 7. Nevertheless, the robots were only allowed to navigate certain corridors. In Figure 8.1 is illustrated the navigable areas where the tests were performed, as well as the SM that all robots own initially.

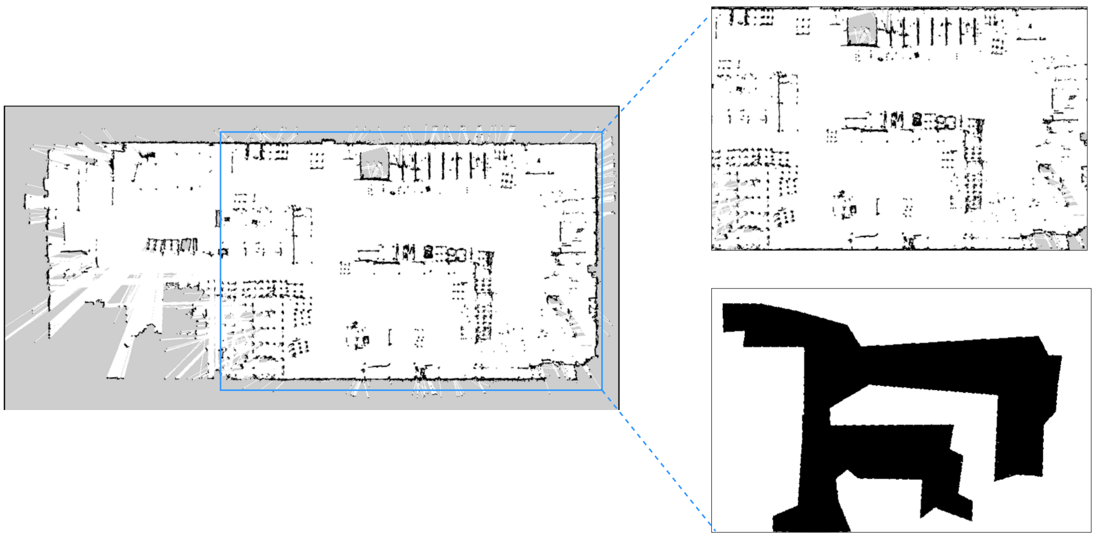
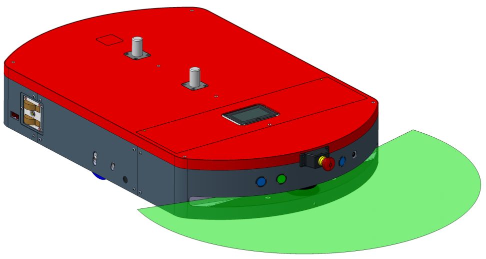
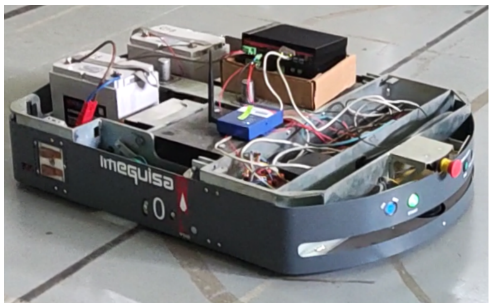


Figure 8.1: Imeguisa's installations and navigable areas. On the bottom right is represented the static map of the navigable corridors.

Figures 8.2a and 8.2b illustrate the representation and a photograph of the real robot used in the experimental tests. The AMR is a differential drive vehicle that can be described by the model presented in Section 3.3. The vehicle's parameters used in the motion planning problem are presented in Table 8.1. These parameters are, respectively, the vehicle's length and width, the distance between both controllable wheels, the limitations in velocity and acceleration of each wheel, the sensor's horizontal range and the safe distance from obstacles.


(a) Representation of the real AMR. The sensor's angular range is illustrated in green.



(b) Real AMR used in the experimental tests.

Figure 8.2: Representation and real autonomous mobile robot used in the experimental tests [1].

Table 8.1: Vehicle parameters used in the experimental tests.

Parameter	h	w	L	v_{max}	a_{max}	β	d_{safe}
Value	$1.20m$	$0.72m$	$0.63m$	$1ms^{-1}$	$0.5ms^{-2}$	$180°$	$0.6m$

A maximum of three AMRs were used in the experimental tests. The decentralized robots communicate using *Wi-Fi* and through a Robot Operating System (ROS) [54] Multimaster package [2]. Additionally, with odometry and the data provided by the horizontal sensor, the vehicle can navigate in the environment. *amcl* [3] and *move_base* [4] packages are used by each AMR for location and navigation, respectively.

As explained in the project's functional architecture (Section 1.3), the Task Manager has the responsibility of deciding which robot is the most apt to execute each task. Also, for the Task Executer, it uses a local planner that constantly computes a path that minimizes the distance to the generated trajectory and sends velocity commands to the robot's motors to follow it. The local planner used in these preliminary tests was the Timed Elastic Band (TEB) [5]. These two modules were implemented by J. Tavares [7].

8.2 Experimental Tests

Each AMR begins at rest in the idle zone. When a robot finishes a task and does not have a new one assigned, it must drive to the idle zone, which is represented as a yellow rectangle in Figure 8.3. A set of four different tasks $\mathcal{S} = \{s^1, s^2, s^3, s^4\}$ aree assigned. Each task has one starting point and, for testing purposes, only one destination. This can be specified for each task as follows:

[1]Imeguisa authorizes the publication of both images in this dissertation.
[2]A. Tiderko. multimaster fkie. ROS wiki, 2022. URL http://wiki.ros.org/multimaster˙fkie.
[3]B. P. Gerkey. amcl. ROS wiki, 2022. URL http://wiki.ros.org/amcl.
[4]E. Marder-Eppstein. move base. ROS wiki, 2022. URL http://wiki.ros.org/move˙base.
[5]C. Rosmann. teb ¨ local planner. ROS wiki, 2022. URL http://wiki.ros.org/teb˙local˙planner.

- **Task 1** (s^1): Task starts at Location 2 and finishes at Location 1.
- **Task 2** (s^2): Task starts at Location 3 and finishes at Location 4.
- **Task 3** (s^3): Task starts at Location 4 and finishes at Location 3.
- **Task 4** (s^4): Task starts at Location 1 and finishes at Location 2.

Locations 1 to 4 are illustrated in Figure 8.3.

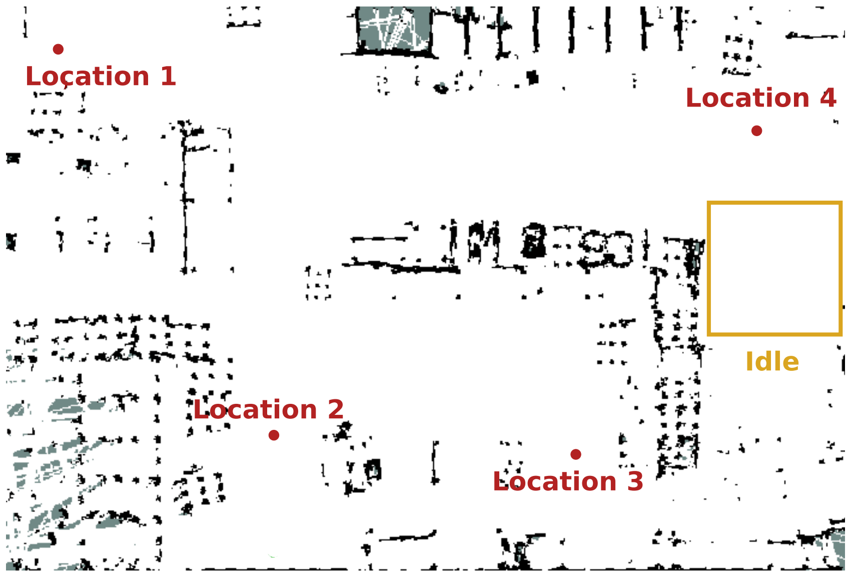


Figure 8.3: Location of starting and ending points of existing tasks (red). Idle zone (yellow), i.e., where a robot must be when no task is assigned to it.

To evaluate the proposed planning techniques integrated with the task assignment strategy, tests were conducted for three scenarios, which are described as follows:

- **Scenario I:** Comprehends a set of two AMRs, R^1 and R^2, to execute task s^1.
- **Scenario II:** Comprehends a set of two AMRs, R^1 and R^2, to execute tasks s^2 and s^3.
- **Scenario III:** Comprehends a set of three AMRs, R^1, R^2 and R^3, to execute tasks s^2, s^3 and s^4.

8.2.1 Scenario I

To better understand Scenario I, the diagram in Figure 8.4 depicts the actions performed along this experimental test. Initially, both robots R^1 and R^2 are in the idle zone (Figure 8.5a). Task s^1 is triggered at $t = 0s$ and an auction is held to determine which robot is executing s^1 (line 1 in Figure 8.4). Each robot, in order to obtain its auction bid, must plan a trajectory from its current position to the starting point of s^1, Location 2. The wining robot, R^1, drives to Location 2 using the trajectory computed for the bid, as illustrated in Figure 8.5b (line 2 in Figure 8.4). In Figure 8.5c, R^1 has already arrived to its destination and plans a new trajectory from the beginning to the end of the assigned task (line 3 in Figure 8.4). Finally, Figure 8.5d illustrates R^1's trajectory to return to the idle zone when the task is completed.

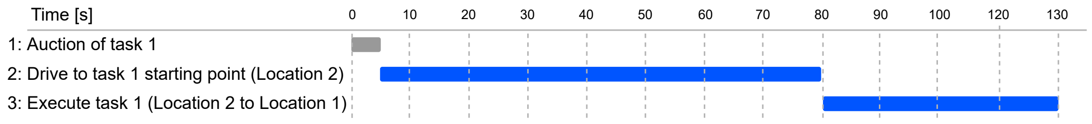
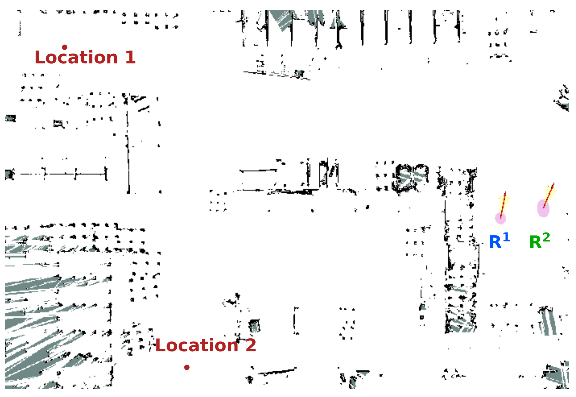
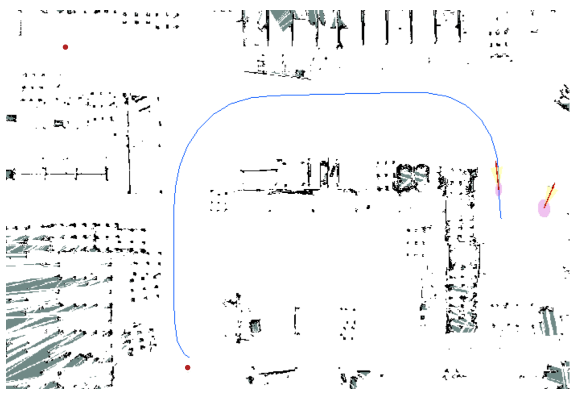


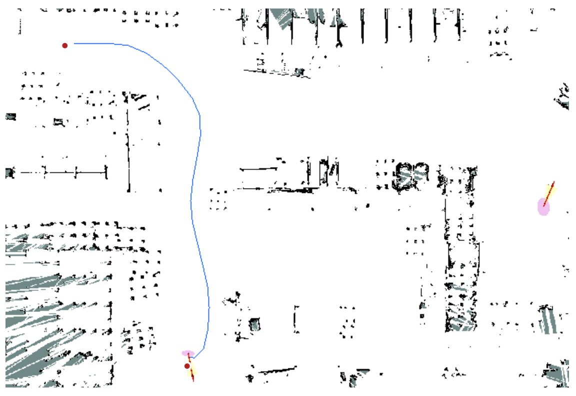
Figure 8.4: Sequential actions performed in Scenario I. Line 1: the auction of task 1, s^1, is triggered. Line 2: the winning robot must drive to the starting point of s^1. Line 3: task s^1 must be performed. The blue lines indicates that vehicle R^1 was chosen to execute s^1.



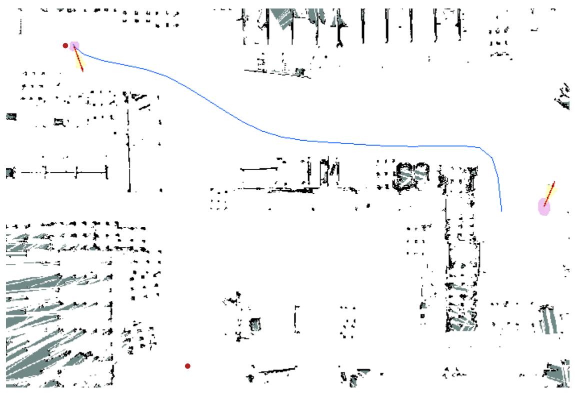
(a) $t = 0s$: s^1 is triggered. Vehicles R^1 (blue) and R^2 (green) are in the idle zone. Red arrows represent the orientation of both vehicles.



(b) $t = 6.54s$: R^1 is allocated to execute task s^1 and begins its journey (blue curve) to the starting point of s^1, Location 2. R^2 is in the idle zone.



(c) $t = 80.57s$: R^1 begins its journey (blue curve) from the starting to the ending point of s^1, Locations 2 and 1, respectively. R^2 is in the idle zone.



(d) $t = 135.42s$: R^1 finishes s^1 and begins a journey (blue curve) to return to the idle zone. R^2 is in the idle zone.

Figure 8.5: Scenario at Imeguisa's installations with a set of two robots to execute task s^1. [6]

[6]youtube video URL: https://youtu.be/BCL9o4V_nM0

8.2.2 Scenario II

The diagram in Figure 8.6 depicts the actions performed in Scenario II. The second scenario begins at $t = 0s$, with task s^2 being triggered (line 1 in Figure 8.6). As shown in Figure 8.7b, R^1 wins s^2's auction and begins its trajectory to the starting point of its allocated task, Location 3 (line 2 in Figure 8.6). Then, s^3's auction is held (line 4 in Figure 8.6). Now, R^1's bid must consider both the time required to complete its current task and the time needed to reach s^3's starting position. Since it is faster for R^2 to travel from the idle zone to Location 4, task s^3 is assigned to R^2. Figure 8.8a depicts the trajectory from R^2's current position to the starting point of s^3, Location 4 (line 5 in Figure 8.6).

In Figure 8.8b, R^2 arrives to Location 4 and plans a new trajectory to Location 3 (line 6 in Figure 8.6). To complete task s^2, R^1 begins its journey to Location 4, as shown in Figure 8.8c (line 3 in Figure 8.6). Figure 8.8d illustrates a situation in which the two vehicles share the same corridor. In fact, the planned trajectories prevent both vehicles from colliding with each other. Also, it is verified that each robot is driving on its right-hand side. Finally, Figures 8.8e and 8.8f show R^2 and R^1 completing their tasks and planning their routes to the idle zone, respectively.

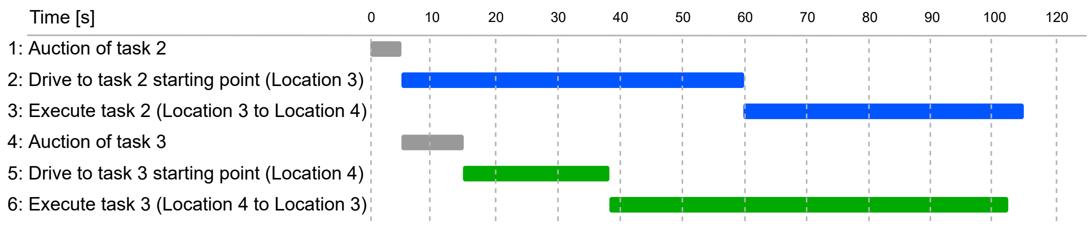
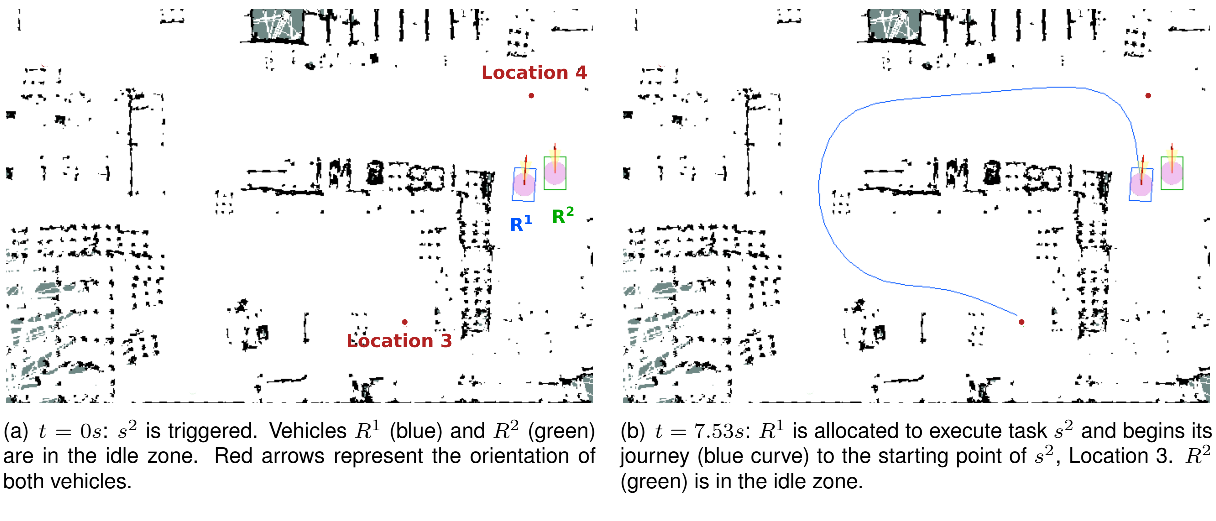


Figure 8.6: Sequential actions performed in Scenario II. Line 1: the auction of task 2, s^2, is triggered. Line 2: the winning robot must drive to the starting point of s^2. Line 3: task s^2 must be performed. The blue lines indicates that vehicle R^1 was chosen to execute s^2 Line 4: the auction of task 3, s^3, is triggered. Line 5: the winning robot must drive to the starting point of s^3. Line 6: task s^3 must be performed. The green lines indicates that vehicle R^2 was chosen to execute s^3.

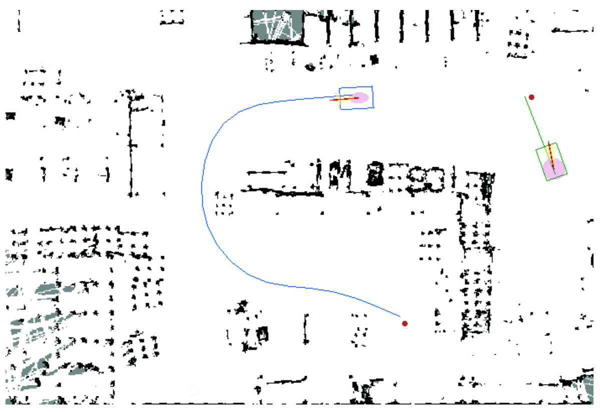


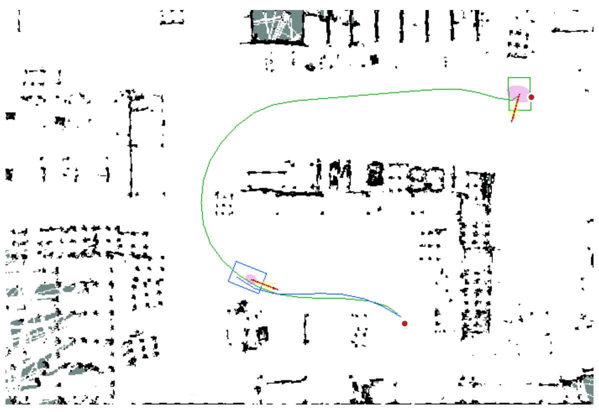
(a) $t = 0s$: s^2 is triggered. Vehicles R^1 (blue) and R^2 (green) are in the idle zone. Red arrows represent the orientation of both vehicles.

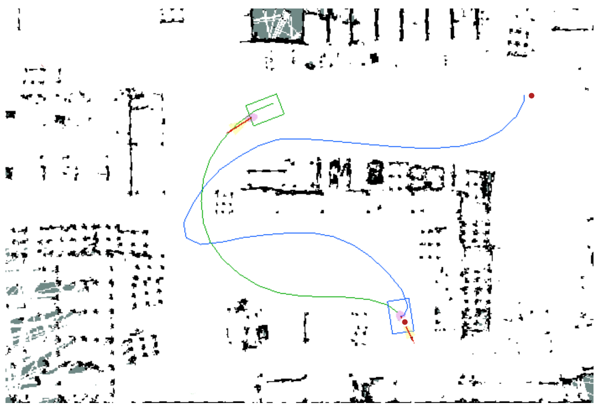
(b) $t = 7.53s$: R^1 is allocated to execute task s^2 and begins its journey (blue curve) to the starting point of s^2, Location 3. R^2 (green) is in the idle zone.

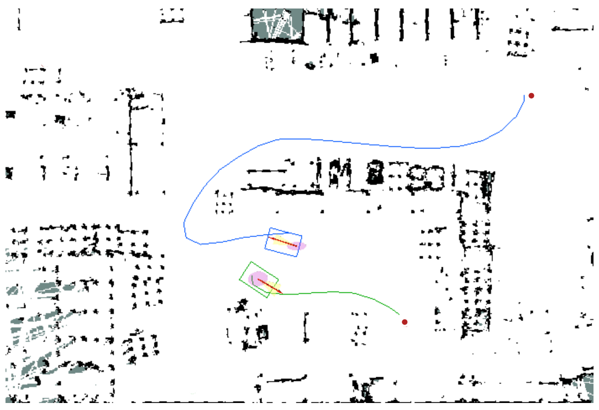
Figure 8.7: Scenario at Imeguisa's installations with a set of two robots to execute tasks s^2 and s^3. [7]

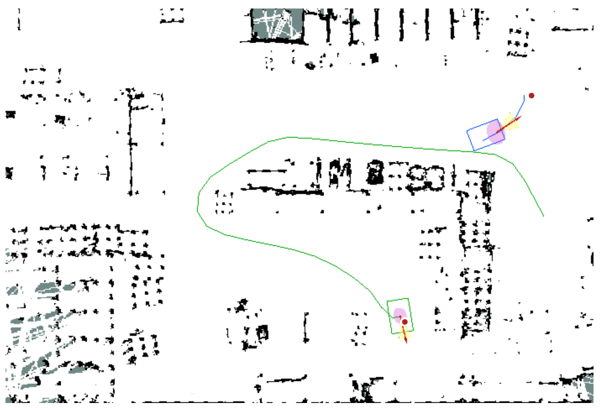
[7]youtube video URL: https://youtu.be/WspBdT7FeT4


(a) $t = 16.50s$: R^2 is allocated to execute task s^3 and begins its journey (green line) to the starting point of s^3, Location 4. R^1 is following the trajectory previously initiated (blue curve).


(b) $t = 38.56s$: R^2 begins its journey (green curve) from the starting to the ending point of s^3, Locations 4 and 3, respectively. R^1 is following the trajectory previously initiated (blue curve).


(c) $t = 59.14s$: R^1 begins its journey (blue curve) from the starting to the ending point of s^2, Locations 3 and 4, respectively. R^2 is following the trajectory previously initiated (green curve).


(d) $t = 74.76s$: R^1 (blue) and R^2 (green) avoid each other. The red arrows representing both vehicles' orientation, show that each vehicle drives on its right-hand side.


(e) $t = 105.00s$: R^2 finishes s^3 and begins a journey (green curve) to return to the idle zone. R^1 is following the trajectory previously initiated (blue curve).

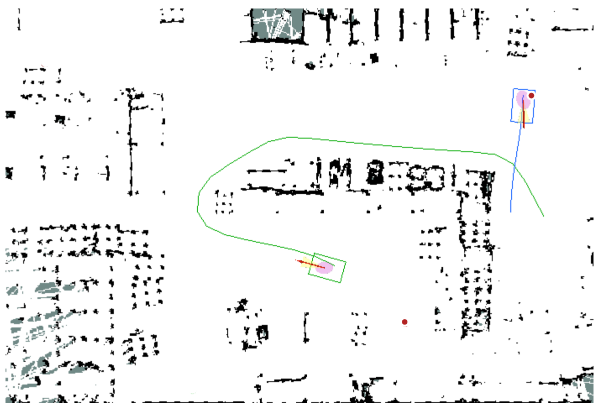

(f) $t = 116.00s$: R^1 finishes s^2 and begins a journey (blue curve) to return to the idle zone. R^2 is following the trajectory previously initiated (green curve).

Figure 8.8: Scenario at Imeguisa's installations with a set of two robots to execute tasks s^2 and s^3 (cont.).

8.2.3 Scenario III

The diagram in Figure 8.9 depicts the actions performed in Scenario III. In the final test, three robots R^1, R^2, and R^3 complete three tasks s^2, s^3, and s^4. The tasks are allocated sequentially. First, s^4 is allocated (line 1 in Figure 8.9), then s^2 (line 4 in Figure 8.9) and , finally, s^3 (line 7 in Figure 8.9). Figure 8.10b shows that task s^4 is assigned to R^1 (line 2 in Figure 8.9). Then, task s^2 is auctioned (line 5 in Figure 8.9). During this auction, the three robots compute their bids to drive to s^2's initial point, Location 3. Even though R^1 is the closest to Location 3, it still has to complete s^4. Thus, R^1's bid must consider finishing its assigned task and, posteriorly, driving to Location 3. On the other hand, R^2 and R^3's bids only consider the time it takes to drive from the idle zone to Location 3. As shown in Figure 8.11a, task s^2 is allocated to R^2.

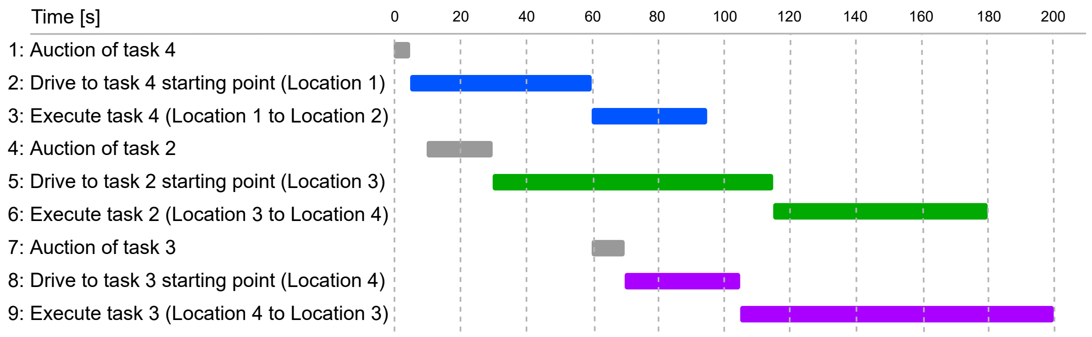
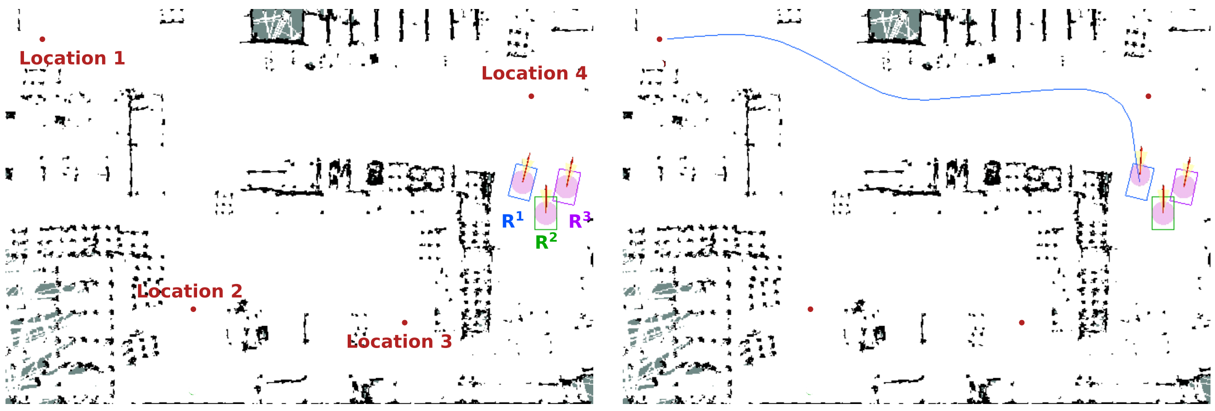


Figure 8.9: Sequential actions performed in Scenario III. Line 1: the auction of task 4, s^4, is triggered. Line 2: the winning robot must drive to the starting point of s^4. Line 3: task s^4 must be performed. The blue lines indicates that vehicle R^1 was chosen to execute s^4 Line 4: the auction of task 2, s^2, is triggered. Line 5: the winning robot must drive to the starting point of s^2. Line 6: task s^2 must be performed. The green lines indicates that vehicle R^2 was chosen to execute s^2. Line 7: the auction of task 3, s^3, is triggered. Line 8: the winning robot must drive to the starting point of s^3. Line 9: task s^3 must be performed. The purple lines indicates that vehicle R^3 was chosen to execute s^3.

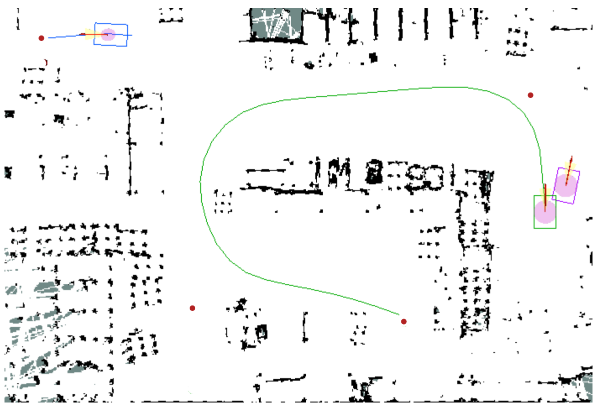


(a) $t = 0s$: s^4 is triggered. Vehicles R^1 (blue), R^2 (green) and R^3 (purple) are in the idle zone. Red arrows represent the orientation of both vehicles.

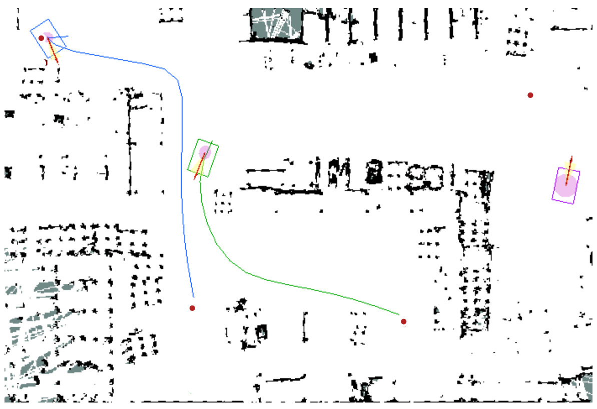
(b) $t = 3.44s$: R^1 is allocated to execute task s^4 and begins its journey (blue curve) to the starting point of s^4, Location 1. R^2 (green) and R^3 (purple) are in the idle zone.

Figure 8.10: Scenario at Imeguisa's installations with a set of three robots to execute tasks s^2, s^3 and s^4. [8]

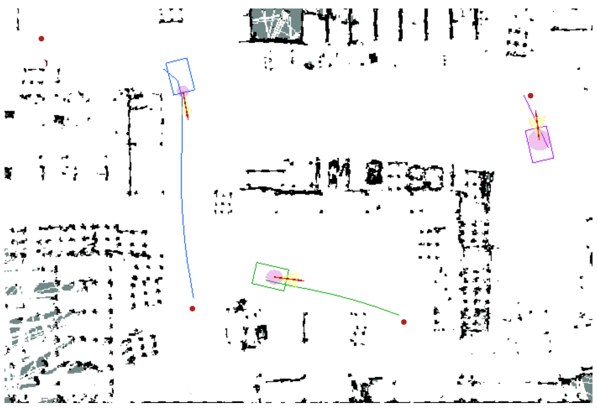
[8]youtube video URL: https://youtu.be/S9P7CLYURyE



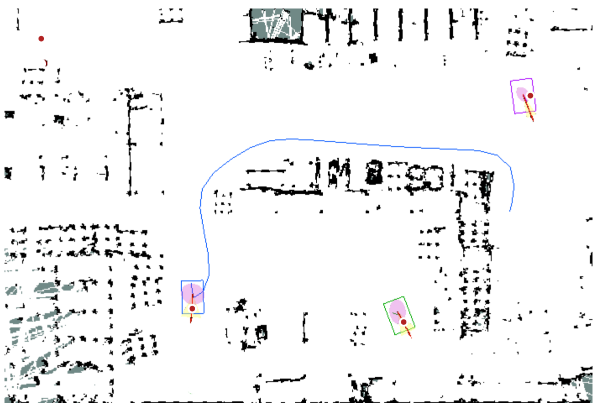
(a) $t = 30.92s$: R^2 is allocated to execute task s^2 and begins its journey (green curve) to the starting point of s^2, Location 3. R^1 is following the trajectory previously initiated (blue curve). R^3 (purple) is in the idle zone.



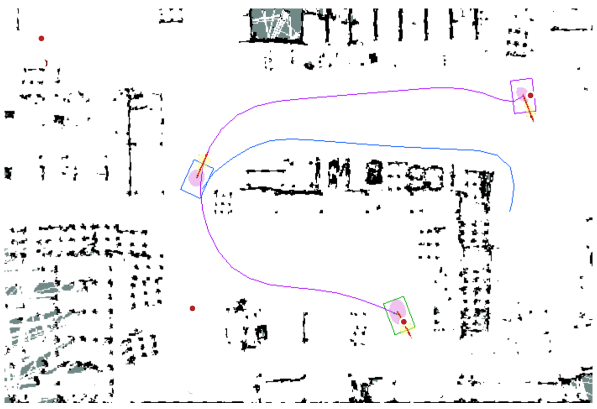
(b) $t = 64.02s$: R^1 begins its journey (blue curve) from the starting to the ending point of s^4, Locations 1 and 2, respectively. R^2 is following the trajectory previously initiated (green curve). R^3 (purple) is in the idle zone.



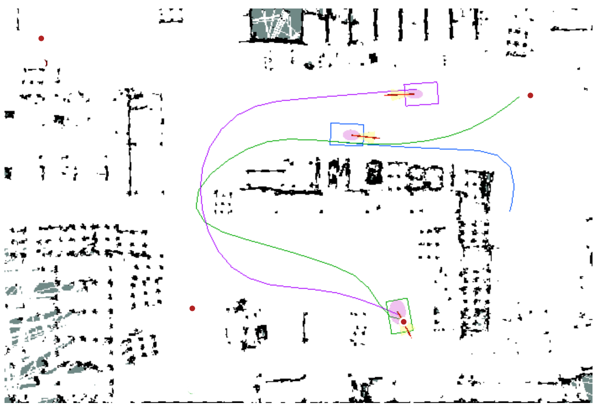
(c) $t = 75.39s$: R^3 is allocated to execute task s^3 and begins its journey (purple curve) to the starting point of s^3, Location 4. R^1 and R^2 are following the trajectories previously initiated (blue and green curves, respectively).



(d) $t = 94.86s$: R^1 finishes s^4 and begins a journey (blue curve) to return to the idle zone. R^2 (green) arrives to Location 3. R^3 (purple) arrives to Location 4.

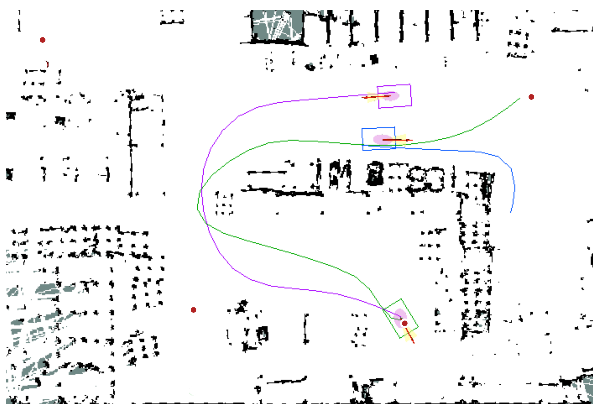


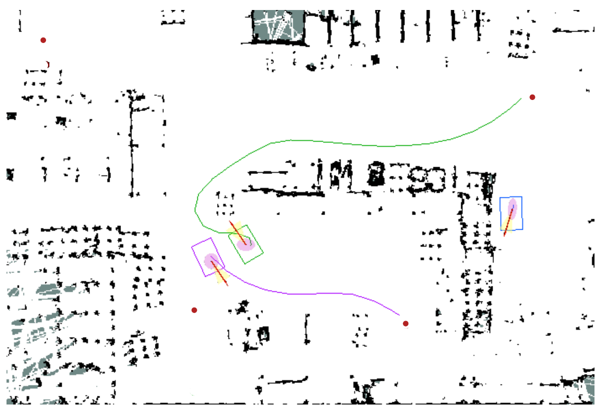
(e) $t = 106.14s$: R^3 begins its journey (purple curve) from the starting to the ending point of s^3, Locations 4 and 3, respectively. R^1 is following the trajectory previously initiated (blue curve). R^2 (green) is at Location 3.

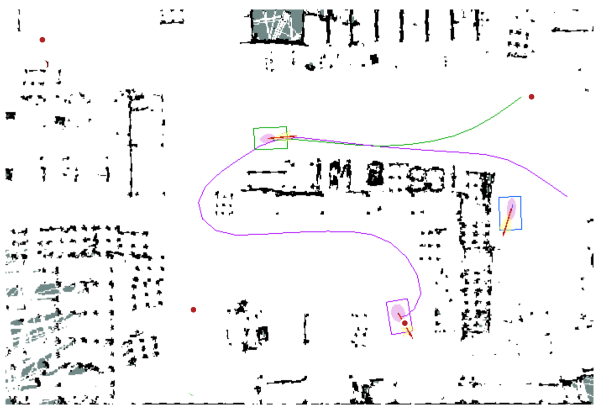


(f) $t = 117.044s$: R^2 begins its journey (green curve) from the starting to the ending point of s^2, Locations 3 and 4, respectively. R^1 and R^3 are following the trajectories previously initiated (blue and purple curves, respectively).

Figure 8.11: Scenario at Imeguisa's installations with a set of three robots to execute tasks s^2, s^3 and s^4 (cont.).


(a) $t = 119.31s$: R^1 (blue) and R^3 (green) avoid each other. The red arrows representing both vehicles' orientation, show that each vehicle drives on its right-hand side.


(b) $t = 140.61s$: R^2 (green) and R^3 (purple) avoid each other. The red arrows representing both vehicles' orientation, show that each vehicle drives on its right-hand side. R^1 arrives to the idle zone.


(c) $t = 203.68s$: R^3 finishes s^3 and begins a journey (purple curve) to return to the idle zone. R^2 is following the trajectory previously initiated (green curve). R^1 (blue) is in the idle zone.

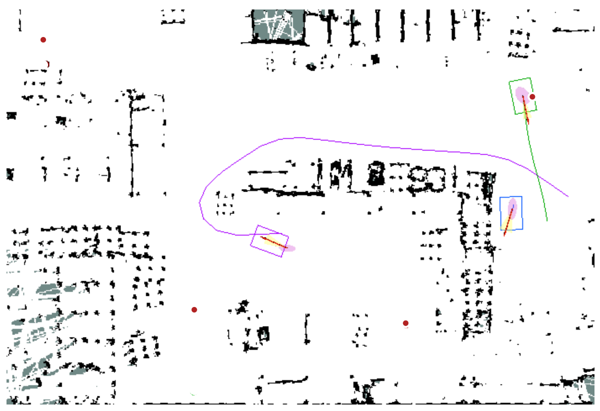

(d) $t = 180.08s$: R^2 finishes s^2 and begins a journey (green curve) to return to the idle zone. R^3 is following the trajectory previously initiated (purple curve). R^1 (blue) is in the idle zone.

Figure 8.12: Scenario at Imeguisa's installations with a set of three robots to execute tasks s^2, s^3 and s^4 (cont.).

Figure 8.11b shows R^1 arriving at s^4's starting point and generating a trajectory to the task's endpoint (line 3 in Figure 8.9). The final task, s^3, is allocated to R^3 (line 8 in Figure 8.9), Figure 8.11c. Figure 8.11d shows R^2 and R^3 arriving to their respective task's starting point. Additionally, R^1 completes its task and plans a route to return to the idle zone. Figures 8.11e and 8.11f show R^3 and R^2 planning their journeys from the starting to the finishing point of their tasks, respectively (lines 9 e 6 in Figure 8.9, respectively).

Figures 8.12a and 8.12b illustrate two situations in which the vehicles drive on their right-hand sides, preventing them from colliding with each other. Finally, Figures 8.12c and 8.12d show R^3 and R^2 completing their tasks and planning their routes to the idle zone, respectively.

8.2.4 Final Remarks

The method proposed was suitable for generating collision-free and feasible trajectories for a fleet of three real AMRs with non-linear constraints in a real environment. Nevertheless, in a real-world industrial setting, the time required to generate each trajectory is a crucial factor that must be considered. When a task is triggered, the algorithm must be able to plan a route to execute it as quickly as possible. In addition, if a recalculation of a trajectory is required due to environmental changes, the robot must not block the corridor while waiting for a new route to be calculated. Moreover, the synchronization between robots is a crucial factor that may be compromised by high computational times.

Therefore, the computational time [9] of the generated trajectories used in the auction bids was gathered for the tests conducted in scenarios similar to those described above. This data was divided into five categories. When a task is auctioned, five possible routes might be planned:

- Route from the idle zone to Location 1.
- Route from the idle zone to Location 2.
- Route from the idle zone to Location 3.
- Route from the idle zone to Location 4.
- The robot has a task already allocated and needs to consider a route from its current position to the endpoint of the respective assigned task and, then, to the startpoint of the auctioned task, which might be Locations 1, 2, 3 or 4.

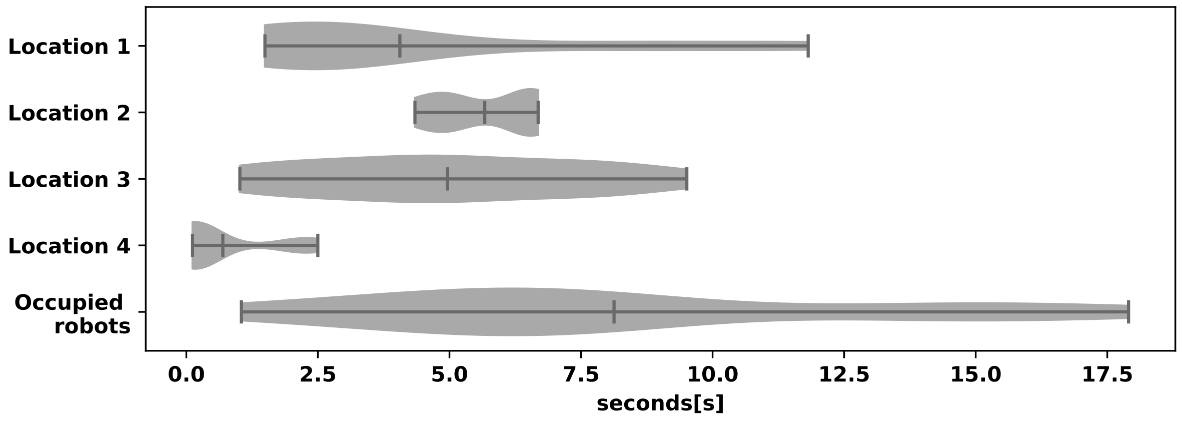


Figure 8.13: Computational time for generating optimal collision-free and feasible trajectories in a real-world scenario.

Figure 8.13 illustrates the collected data as violin plots. It can be verified that planning a trajectory from the idle zone to Location 4 requires the least amount of time, as the shortest path is generated. However, when a robot is occupied, takes longer to generate its bid since it must consider two routes: from its current location to the endpoint of the assigned task and then to the startpoint of the auctioned task. In general, the computational time is still considerable. In future work, it will be essential to restructure the proposed method to reduce its computational time.

[9]Each robot has an onboard computer with an Intel Core i5-1145G7E Processor 2.60GHz, 8GB of RAM. The operative system used is Ubuntu 20.04.1.

Chapter 9

Conclusions and Future Work

The work developed throughout this dissertation is aimed at creating a framework that can generate collision-free and feasible trajectories for a fleet of autonomous vehicles with non-linear constraints in obstacle-populated scenarios. In particular, the main purpose was to implement the proposed algorithm in a fleet of AMRs that will operate in Volkswagen Autoeuropa facilities.

The motion planning problem was decoupled into two modules. The first module performs a sampling-based Kinodynamic RRT trajectory search to find a collision-free route, while the second one uses trapezoidal direct collocation to optimize the previous solution into a smooth and collision-free trajectory. This formulation presented only one restriction in the optimization problem regarding static and moving obstacle avoidance. As the main novelty of this dissertation, a set of signed distance fields were used to describe the environment and the dynamic obstacles through time. This renders the solver's performance independent of the number of agents.

The proposed method was initially validated for a six-vehicle problem. The algorithm scalability was then evaluated. Finally, a fifty-agent motion optimization problem was analysed and it was concluded that the increase in computational complexity with the number of vehicles occurs at a linear rate less than unitary.

Some reformulations were required to implement the designed motion planning in a real-world scenario. To attract vehicles to the right side of the factory's corridors, a set of signed distance fields was used to define a measure of violation to be considered in the objective function of the optimization problem. Additionally, an algorithm was presented that allows all vehicles to have an up-to-date map of their surroundings whenever other robots detect changes in the environment.

Finally, preliminary experimental tests for the AGiLE project were performed at Imeguisa's facilities. Overall, the proposed framework provided a satisfactory solution to the real-world trajectory planning problem. However, the system requires a significant amount of computation time in order to generate a feasible trajectory.

Concerns must be addressed prior to advancing to the next phase of testing in more complex scenarios.

As previously stated, it is crucial to reduce the computation time required to generate a collision-free feasible trajectory. Adjustments on the Trajectory Planning module, i.e., on the Kinodynamic RRT, could be one solution. Since the majority of the journeys already have predefined initial and destination coordinates, a RRT tree could be precomputed on every known location where a trajectory might be initiated. Thus, RRT trees can be generated only once for each location of a task's starting or ending point, ensuring that it would cover the entire area. Then, subsequent vehicles initiating a route at those locations could use those precomputed trees, thereby reducing the computation time. Another possibility is related with the Trajectory Optimization module. The size of each signed distance field corresponds to the size of the map used. Increasing the map's resolution will result in the use of smaller matrices, which could reduce the computation time.

Some additional improvements to the implemented Kinodynamic RRT should to be considered. Frequently, this randomized trajectory planner delivers a first initial trajectory of such poor quality that not even the optimizer can completely smooth it, leaving some unnecessary curves and manoeuvrers. Therefore, it is essential to attempt to enhance the quality of the Kinodynamic RRT solution.

The importance of robot synchronization has already been discussed regarding robots' computation times. Nevertheless, this synchronization is frequently jeopardize by the local planner (in the experimental tests the TEB was used), which is responsible for ensuring that the provided trajectory is correctly followed by the real AMR. This local planner is only prepared to accept paths and not trajectories. Thus, it would only consider coordinates and not time or velocity commands. This may result in a deviation between the planned trajectory and the executed trajectory. This makes synchronization between all robots more difficult, as none of them knows precisely where the others are at any given moment. As future work, it would be advantageous to adjust or create a new local planner that is able to adhere to a given trajectory and not just a path.

Bibliography

[1] J. Carvalho. *Logística*. Sílabo, 3rd edition, 2002. isbn : 9789726182795.

[2] M. Hompel, T. Schmidt, L. Nagel, and R. Jünemann. *Material Flow Systems*. Springer, 3rd edition, 2007. isbn : 9783540732365.

[3] F. Kendoul. Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. In *Journal of Field Robotics*, volume 29, pages 315–378. Wiley Online Library, 2012. doi : 10.1002/rob.20414.

[4] R. Saber, W. Dunbar, and R. Murray. Cooperative control of multi-vehicle systems using cost graphs and optimization. In *American Control Conference*, volume 3, pages 2217–2222. IEEE, 2003. doi : 10.1109/ACC.2003.1243403.

[5] W. Meng, Z. He, R. Su, A. Shehabinia, L. Lin, R. Teo, and L. Xie. Decentralized control of multi-UAVs for target search, tasking and tracking. In *IFAC Proceedings Volumes*, volume 47, pages 10048–10053. Elsevier, 2014. doi : 10.3182/20140824-6-ZA-1003.02665.

[6] Y. Zhao, Z. Zheng, and Y. Liu. Survey on computational-intelligence-based UAV path planning. In *Knowledge-Based Systems*, volume 158, pages 54–64. Elsevier, 2018. doi : 10.1016/0021-9991(88)90002-2.

[7] J. Tavares. Decentralized market-based task alloction for a fleet of industrial mobile robots (submitted). Master's dissertation, Instituto Superior Técnico, 2022.

[8] A. Wolek and C. Woolsey. Model-based path planning. In *Sensing and Control for Autonomous Vehicles*, page 183–206. Springer, 2017. doi : 10.1007/978-3-319-55372-69.

[9] S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

[10] J. C. Latombe. *Robot Motion Planning*, chapter 2-3. Kluwer Academic Publishers, USA, 1991.

[11] E. W. Dijkstra. A note on two problems in connexion with graphs. In *Numerische Mathematik*, volume 1, pages 269–271. Springer, December 1959. doi : 10.1007/bf01386390.

[12] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. In *IEEE Transactions on Systems Science and Cybernetics*, July 1968. doi : 10.1109/TSSC.1968.300136.

[13] A. Stentz. Optimal and efficient path planning for partially known environments. In *Intelligent Unmanned Ground Vehicles*, pages 203–220. Springer, 1997. doi : 10.1007/978-1-4615-6325-9_11.

[14] L. Kavraki, P. Svestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *IEEE Transactions on Robotics and Automation*, volume 12, pages 566 – 580, September 1996. doi : 10.1109/70.508439.

[15] S. M. LaValle, J. Kuffner, B. Donald, et al. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and computational robotics: new directions*, volume 5, pages 293–308, 2001.

[16] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. In *The International Journal of Robotics Research*, volume 30, pages 846–894, 2011. doi : 10.1177/0278364911406761.

[17] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *IEEE International Conference on Robotics and Automation*, 2:500–505, 1985. doi : 10.1109/ROBOT.1985.1087247.

[18] J.O. Kim and P.K. Khosla. Real-time obstacle avoidance using harmonic potential functions. In *IEEE Transactions on Robotics and Automation*, volume 8, pages 338–349, 1992. doi : 10.1109/70.143352.

[19] E. Rimon and D.E. Koditschek. Exact robot navigation using artificial potential functions. In *IEEE Transactions on Robotics and Automation*, volume 8, pages 501–518, 1992. doi : 10.1109/70.163777.

[20] S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations. In *Journal of Computational Physics*, volume 79, pages 12–49, 1988. doi : 10.1016/0021-9991(88)90002-2.

[21] A. Valero-Gomez, J. V. Gomez, S. Garrido, and L. Moreno. The path to efficiency: Fast marching method for safer, more efficient mobile robot trajectories. In *IEEE Robotics Automation Magazine*, volume 20, pages 111–120, 2013. doi: 10.1109/MRA.2013.2248309.

[22] S. Garrido, L. Moreno, D. Blanco, and F. Martin. FM2: A real-time fast marching sensor-based motion planner. In *IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 1–6. IEEE, 2007. doi : 10.1109/AIM.2007.4412505.

[23] M. Kelly. An introduction to trajectory optimization: How to do your own direct collocation. In *SIAM Review*, volume 59, pages 849–904. SIAM, 2017. doi: 10.1137/16M1062569.

[24] A. Chamseddine, Y. Zhang, C. A. Rabbath, C. Join, and D. Theilliol. Flatness-based trajectory planning/replanning for a quadrotor unmanned aerial vehicle. In *IEEE Transactions on Aerospace and Electronic Systems*, volume 48, pages 2832–2848, 2012. doi: 10.1109/TAES.2012.6324664.

[25] A. Richards and J. P. How. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In *IEEE American Control Conference*, volume 3, pages 1936–1941. IEEE, 2002. doi: 10.1109/ACC.2002.1023918.

[26] E. Masehian and G. Habibi. Robot path planning in 3d space using binary integer programming. In *International Journal of Computer and Information Engineering*, volume 1, pages 1255 – 1260. World Academy of Science, Engineering and Technology, 2007. doi: 10.5281/zenodo.1072461.

[27] J. T. Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. Society for Industrial and Applied Mathematics, 2nd edition, 2010.

[28] J. T. Betts. Survey of numerical methods for trajectory optimization. In *Journal of guidance, control, and dynamics*, volume 21, pages 193–207, 1998. doi: 10.2514/2.4231.

[29] D. Mellinger, A. Kushleyev, and V. Kumar. Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. In *IEEE international conference on robotics and automation*, pages 477–483. IEEE, 2012. doi: 10.1109/ICRA.2012.6225009.

[30] F. Augugliaro, A. P. Schoellig, and R. D'Andrea. Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1917–1922, 2012. doi: 10.1109/IROS.2012.6385823.

[31] J. Berg, M. Lin, and D. Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *IEEE International Conference on Robotics and Automation*, pages 1928–1935. IEEE, 2008. doi: 10.1109/ROBOT.2008.4543489.

[32] J. Berg, S. J. Guy, M. Lin, and D. Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011. doi: 10.1007/978-3-642-19457-3_1.

[33] D. Zhou, Z. Wang, S. Bandyopadhyay, and M. Schwager. Fast, on-line collision avoidance for dynamic vehicles using buffered voronoi cells. In *IEEE Robotics and Automation Letters*, volume 2, pages 1047–1054. IEEE, 2017. doi: 10.1109/LRA.2017.2656241.

[34] C. E. Luis and A. P. Schoellig. Trajectory generation for multiagent point-to-point transitions via distributed model predictive control. In *IEEE Robotics and Automation Letters*, volume 4, pages 375–382. IEEE, 2019. doi: 10.1109/LRA.2018.2890572.

[35] M. Čáp, P. Novák, A. Kleiner, and M. Selecký. Prioritized planning algorithms for trajectory coordination of multiple mobile robots. In *IEEE Transactions on Automation Science and Engineering*, volume 12, pages 835–849. IEEE, 2015. doi: 10.1109/TASE.2015.2445780.

[36] Y. Chen, M. Cutler, and J. P. How. Decoupled multiagent path planning via incremental sequential convex programming. In *IEEE International Conference on Robotics and Automation*, pages 5954–5961. IEEE, 2015. doi: 10.1109/ICRA.2015.7140034.

[37] D. R. Robinson, R. T. Mar, K. Estabridis, and G. Hewer. An efficient algorithm for optimal trajectory generation for heterogeneous multi-agent systems in non-convex environments. In *IEEE Robotics and Automation Letters*, volume 3, pages 1215–1222. IEEE, 2018. doi: 10.1109/LRA.2018.2794582.

[38] S. Yu, C. Fu, A. K Gostar, and M. Hu. A review on map-merging methods for typical map types in multiple-ground-robot slam solutions. In *Sensors*, volume 20, page 6988, 2020. doi: 10.3390/s20236988.

[39] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun. Collaborative multi-robot exploration. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 476–481. IEEE, 2000. doi: 10.1109/ROBOT.2000.844100.

[40] S. Carpin. Fast and accurate map merging for multi-robot systems. In *Autonomous robots*, volume 25, pages 305–316. Springer, 2008. doi: 10.1007/s10514-008-9097-4.

[41] W. Huang and K. Beevers. Topological map merging. In *The International Journal of Robotics Research*, volume 24, pages 601–613. SAGE Publications, 2005.

[42] F. Abrate, B. Bona, M. Indri, S. Rosa, and F. Tibaldi. Multi-robot map updating in dynamic environments. In *Distributed Autonomous Robotic Systems*, pages 147–160. Springer, 2013.

[43] A. Kleiner, D. Sun, and D. Meyer-Delius. ARMO: Adaptive roadmap optimization for large robot teams. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3276–3282. IEEE, 2011. doi: 10.1109/IROS.2011.6094734.

[44] N. Zoghby, V. Cherfaoui, and T. Denoeux. Evidential distributed dynamic map for cooperative perception in vanets. In *IEEE Intelligent Vehicles Symposium Proceedings*, pages 1421–1426. IEEE, 2014. doi: 10.1109/IVS.2014.6856550.

[45] X. Zhang, A. Liniger, and F. Borrelli. Optimization-based collision avoidance. In *IEEE Transactions on Control Systems Technology*, volume 29, pages 972–983. IEEE, 2021. doi: 10.1109/TCST.2019.2949540.

[46] K. Bergman. *Exploiting Direct Optimal Control for Motion Planning in Unstructured Environments*. PhD thesis, Linkoping University Electronic Press, 2021.

[47] S. M. LaValle and J. Kuffner Jr. Randomized kinodynamic planning. In *The international journal of robotics research*, volume 20, pages 378–400. SAGE Publications, 2001. doi: doi.org/10.1177/027836401220674.

[48] T. Chan and Wei Z. Level set based shape prior segmentation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 1164–1170. IEEE, 2005. doi: 10.1109/CVPR.2005.212.

[49] D. Kraft. Algorithm 733: TOMP–fortran modules for optimal control calculations. In *ACM Transactions on Mathematical Software*, volume 20, page 262–281. Association for Computing Machinery, September 1994. doi: 10.1145/192115.192124.

[50] D. Kraft. *A software package for sequential quadratic programming*. Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt, 1988.

[51] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. In *Mathematical programming*, volume 106, pages 25–57. Springer, 2006. doi: 10.1007/s10107-004-0559-y.

[52] P. Virtanen et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. In *Nature Methods*, volume 17, pages 261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[53] J. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl. CasADi – A software framework for nonlinear optimization and optimal control. In *Mathematical Programming Computation*, volume 11, pages 1–36. Springer, 2019. doi: 10.1007/s12532-018-0139-4.

[54] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.