DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING, SECOND CYCLE, 30 CREDITS

STOCKHOLM, SWEDEN 2020

Multi-Agent Motion Planning with Signal Temporal Logic Constraints

SOPHIE TABOADA

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Multi-Agent Motion Planning with Signal Temporal Logic Constraints

Thesis Report

SOPHIE TABOADA

Master in Electrical and Computer Engineering
Date: November 23, 2020
Supervisor: Fernando dos Santos Barbosa & Pedro Lima
Examiner: Jana Tumová
School of Electrical Engineering and Computer Science

Abstract

Motion planning algorithms allow us to define a sequence of configurations to guide robots from a starting point to an ending goal while considering the environment's and the robot's constraints. As all robots and circumstances are different, motion planning can be adapted to fit into the system's specifications and the user's preferences. Temporal Logic (TL) has been used to enable the implementation of more complex missions. In this work, we are interested in using TL to establish the affiliation between robots in a multi-robot system, as well as their affiliation with features in the workspace. More specifically, Signal Temporal Logic (STL) is used to guide motion planning into respecting certain preferences linked to the robot's motion behavior. In fact, user's preferences are translated into STL formulas, that need to be respected by the motion planning. To achieve this, RRT* sampling-based algorithm is used to study the free space and to identify the best trajectory with the help of a cost analysis of all possible trajectories found. Here, RRT* is adapted to fit into multi-robot systems and to allow the simultaneous planning of trajectories for multiple robots. The robustness metric of STL quantifies the respect trajectories have for STL formulas and influences the cost function of the RRT*. The impact the robustness has on the cost function is responsible for the selection of trajectories with more respect for the STL formulas.

The proposed multi-agent motion planning is tested in simulations with environments containing multiple obstacles and robots. To demonstrate the impact STL has on motion planning, a comparison is made between the trajectories extracted with and without the use of STL. These simulations include specific scenarios and different numbers of robots to test the developed algorithm. They deliver asymptotically optimal solutions. Finally, we conduct some hardware experiments up until four robots to present how the developed motion planning can be implemented in real life.

Summanfattning

Rörelseplaneringsalgoritmer låter oss definiera en sekvens av konfigurationer för att guida robotar från en startposition till en slutposition medan vi tar hänsyn till robotens och miljöns begränsningar. Eftersom alla robotar och omständigheter är olika kan rörelseplanering anpassas för att passa systemets specifikationer och användarens preferenser. Temporal Logik (TL) har använts för att möjliggöra implementationer av mer komplexa uppdrag. I detta arbete är vi intresserade av att använda TL för att fastställa anslutningen mellan robotar i ett multirobotsystem, samt mellan dessa robotar och egenskaper i deras arbetsmiljö. Mer specifikt används signaltemporär logik (eng: Signal Temporal Logic) (STL) för att anpassa rörelseplanering till att respektera vissa preferenser länkade till robotens rörelsebeteende. Faktum är att användarpreferenser översätts till STL-formler som behöver respekteras av rörelseplaneringen. För att uppnå detta används den samplingsbaserade algoritmen RRT* för att studera den fria ytan och för att identifiera den bästa rörelsebanan med hjälp av en kostanalys av alla funna möjliga rörelsebanor. Här anpassas RRT* för multirobotsystem och för att tillåta planering av rörelsebanor för flera robotar samtidigt. Robushetsmåttet för STL kvantifierar respekten som banorna har för STL-formler och påverkar RRT*: s kostnadsfunktion. Påverkan som robustheten har på kostfunktionen är ansvarig för valet av rörelsebanor som till högre grad respekterar STL-formlerna.

Den föreslagna röresleplaneringen för flera agenter (eng: *multi-agent*) testas i simuleringar av miljöer med flera hinder och robotar. För att demonstrera vilken inverkan STL har på rörelseplanering görs en jämförelse mellan rörelsebanor som ges med och utan användning av STL. Dessa simuleringar inkluderar specifika scenarion och olika antal robotar för att testa den utvecklade algoritmen. De levererar asymptotiskt optimala lösningar. Slutligen genomför vi hårdvaruexperiment upp till och med fyra robotar för att presentera hur den framtagna rörelseplaneringsalgoritmen kan implementeras i verkligheten.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Fernando dos Santos Barbosa, and to my examiner, Jana Tumová, for providing me with an incredibly welcoming experience at the Robotic Perceptions and Learning (RPL) division at KTH University. Their continuous guidance and enthusiasm were fundamental for the accomplishment of this research and thesis work. I would also like to thank Felix Büttner for his contribution in the real-life implementation of this project and for his determination to make it possible. Finally, I would like to thank my loving and supporting friends and family, for always motivating me in accomplishing my goals.

Contents

List of Abbreviations

B-RRT* Bidirectional Rapidly-Exploring Random Tree Star

CF Crazyflie

G-RRT* Graph Rapidly-Exploring Random Tree Star

IB-RRT* Intelligent Bidirectional Rapidly-Exploring Random Tree Star

i-RRT* informed Rapidly-Exploring Random Tree Star

LTL Linear Temporal Logic

MA-RRT* Multi-Agent Rapidly-Exploring Random Tree Star

NBA Non-Deterministic Büchi Automaton

PRM Probabilistic Road Maps

RRT Rapidly-Exploring Random Tree

RRT* Rapidly-Exploring Random Tree Star

STL Signal Temporal Logic

TL Temporal Logic

T-RRT* Transition Rapidly-Exploring Random Tree Star

UAV Unmanned Aerial Vehicles

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

One of the cutting-edge research directions in the field of Robotics and Intelligent Systems today focuses on the study of motion planning algorithms for autonomous vehicles. In fact, since it has been a core topic for the past decades, many algorithms have been developed and extended to improve trajectories [1] [2]. The fundamental concept of these motion planning algorithms is to determine possible end-to-end trajectories by controlling the agent position through time and space. They use collision avoidance algorithms to respect the environment's characteristics [3]. Indeed, trajectories are planned carefully so they do not collide with the obstacles or the boundaries of the robot's surroundings. Moreover, motion planning can be customized according to the task the robot has to complete. For instance, some restrictions concerning speed limits or minimum and maximum distances to obstacles can be added to influence the robot's behaviour.
Determining the best trajectory among an infinite amount of possibilities is usually done by analysing trajectory performance (time and length) and the preferences imposed by the user. Standard motion planning algorithms select the trajectories without collisions and with shortest lengths [1] [4]. When additional preferences have to be taken into account for trajectory selection, a study concerning the respect of these preferences is done for each trajectory found. As a result, it chooses a trajectory that best fits this criteria. Temporal Logic (TL) is used in motion planning to establish temporal and/or spacial rules in robot's trajectories. It can define, for instance, the importance, frequency and time of a task execution with the help of multiple operators.

In this thesis work, we use an extension of TL named Signal Temporal Logic (STL). In contrast with TL, which returns Boolean values "True" or "False" whether the trajectory is satisfying the preferences or not, STL quantifies the trajectory satisfaction

over the user's preferences. The quantification of satisfaction or violation of these preferences enables the algorithm to permit some violations if it proves to be more advantageous for the global trajectory. These preferences are translated into formulas that define time and space restrictions concerning the robots' motion behaviour. For instance, STL enables the possibility to determine how far or how close the robot should stay from obstacles [5]. By quantifying the level of satisfaction for user's preferences we can influence path selection in order to choose a trajectory that satisfies these preferences. It is one of the main focuses of this work.

When motion planning algorithms started being brought to light, the curiosity of combining several robots in a same motion planning emerged, originating in what is called today: Multi-agent Motion Planning [6] [7]. In fact, combining simultaneous trajectories can be very useful for the division of tasks between multiple robots. Ideally, it allows a more effective conclusion of tasks compared to when only one robot is responsible for them. However, it can also be quite tricky to plan non-conflicting and simultaneous trajectories for multiple robots through time and space. They might as well result in longer, individual trajectories, both in time and in space to avoid collisions. As a result of preventing robots from colliding with each other, the trajectories might become less efficient.

1.2 Scope

There are many applications for multi-agent motion planning which might involve problems such as task execution, monitoring or supervision, object detection, etc. It is then necessary to establish a motion planning for several robots, which safely prevents them from colliding into one another and with their surroundings.

In this work, a multi-agent system of quadcopter vehicles (UAVs) will be used to monitor different rooms in a known space containing obstacles. The robots' trajectories will need to take into consideration certain rules which will be proposed by STL formulas. These rules imply respecting minimum and maximum distances between robots and between robots and the environment.

The aim of this thesis is therefore, to expand the proposed motion-planning algorithm described in *Barbosa et al., "Guiding Autonomous Exploration With Signal Temporal Logic", 2019* [5] for multi-agent systems and experimentally evaluate its performance with simulations and hardware material. The multi-agent motion planning are inspired in the algorithm elaborated in Kantaros Y. and M. Zavlanos M.'s work: *"Temporal Logic Optimal Control for Large-Scale Multi-Robot Systems: 10400 States and Beyond", 2018* and instead of using Linear Temporal Logic specifications we are using STL specifications to influence the motion planning. Since STL can return the quantification of respect or violation of the preferences for each path, it can be tra-

duced into a gain. This gain is then used to determine the costs of the corresponding paths. It allows the algorithm to add a notion of robustness of STL formulas over the trajectories. This facilitates the trajectory selection process into picking the paths with smaller costs and with more respect for the STL specifications. Another objective of this thesis work, is to determine which motion preferences, defined by STL formulas, can be more advantageous in multi-agent systems situations and if they present alternative results to multi-agent motion planning without STL specifications that can be more useful for specific circumstances.

1.2.1 Problem Formulation

What is expected for this work is to see how the motion planning of multi-agents differs when STL preferences are used to constrain their motion behaviour. Therefore, the research question to be answered by this thesis work is:

- How do trajectories of multi-agent systems differ when Signal Temporal Logic formulas are used to constraint their motion's behaviour?
- How is the performance of this trajectories comparing to trajectories planned without STL? What are the benefits?

We intent from this work to explore the utilities of STL preferences in constraining the motion planning of multi-agent systems. The hypothesis is that STL preferences modify the results of trajectory planning by ensuring user preferences are taken into account. It should show trajectories will only break the preferences if there is no other possibility and try to minimize the costs while doing it. When several robots are involved, using STL should reinforce the prevention of robot collisions by adding safety precautions to their trajectories. STL should enhance the predictability of robot's behaviours when adding extra specifications to their trajectories.

1.3 Outline

In Chapter 2, a background about relevant topics like Temporal Logic and Motion planning Algorithms in this work is explained. We introduce a study about Related Work of already-existent researches about the relevant algorithms of this thesis. In Chapter 3, Temporal Logic concepts relevant for the motion-planning algorithm are brought up. The theory behind Signal Temporal Logic formulas and their construction is introduced. A description of the motion planning algorithm: Multi-agent Random Rapidly-Exploring Tree Star (MA-RRT*) used throughout this thesis is introduced, ranging from simple description of RRT to more advanced algorithm such as RRT*

and then MA-RRT* Since different strategies were used to solve this problem, a description of each will be presented. In Chapter 4, different tests and simulations are carried out and presented by modifying the values of key parameters and comparing the results. In Chapter 5, the results are evaluated with other algorithm alternatives In Chapter 6, hardware experiments are conducted. In chapter 7, we develop on the sustainability and ethical implications of this work. In Chapter 8, future work is elaborated. Finally, in Chapter 9, conclusions are drawn regarding both the obtained results and the comparisons between using and not using STL constraints.

Chapter 2

Background

2.1 Background

2.1.1 Motion Planning Algorithms

A motion planning algorithm focuses on finding a trajectory for a robot to travel from a starting point A to a goal point B, while avoiding obstacles along the way. There are several algorithms approaching this problem, which differ depending on the objective of the motion, to the information available about the environment and to the robot's available utilities, namely sensors, cameras, etc.

On-line and Off-line Planning

In non-real-time (off-line) trajectory planning, motion planning algorithms build trajectories before the motion begins. Meanwhile in real-time (on-line) trajectory planning [8], they can plan the trajectories while the robots are moving in the environment. According to [9], off-line planners are mostly useful for repeatable tasks such as surveillance or many industrial applications, where the environment characteristics are known and the trajectory needs to be optimal. Therefore, off-line trajectory planning is used for motion in environments where the obstacles' position is static or predictable. On the other hand, on-line planners are prepared to deal with unexpected events and can find alternatives or apply corrections to their trajectories. Consequently, the task execution is delayed due to the computational time it takes to decide how to proceed.
To simplify the study of STL constraints in multi-agent systems, this works considers an entirely known workspace with static obstacles. Furthermore, as the robot's trajectories will be planned simultaneously for all robots, a continuous evaluation of robot's free space will be executed to identify whether inter-robot collisions are likely to happen or not.

Sampling-based Algorithms

Early motion planners like cell decomposition methods and potential fields have shown to not be scalable enough when the size and dimension of the environment is too big or when the number of obstacles involved is too high. In [2], Sertac Karaman and Emilio Frazzoli explain the benefits of working with a well-known class of motion planners: sampling-based [2] [10] [11]. In these planners, there is no need to explicitly represent the workspace and its boundaries. When given details about the initial and goal positions, sampling-based motion planning algorithms return a collision free trajectory. To do so, the samples are firstly selected randomly in the configuration space. Then, if a sample corresponds to an obstacle configuration, it is discarded. Otherwise the sample is kept for record. In other words, the algorithm iteratively selects possible intermediate feasible trajectories that do not collide with obstacles and adds them to a road map or a tree until it reaches the final goal destination. This results in minor computational weight and more effectiveness for motion planning in high-dimensional spaces. Two sampling-based motion planning algorithms that have been highlighted in the past years are Probabilistic Road Maps (PRM) [12] and Rapidly-exploring Random Trees (RRT) [12] [13]. Although both of these approaches use the same sampling method, the graph joining the paths extracted from the samples is not built the same way. The PRM method has two main, and not necessarily sequential, phases. They include the road map construction with nodes which are collision free configurations, and creation phase where a path is found from the initial end to the goal end. RRT generates open-loop trajectories for non-linear systems and is meant for motion planning in high-dimensional spaces presenting obstacles. In [12] a comparison is made between PRM and RRT where it is possible to conclude the computational time is considerably bigger for PRM than for RRT. Indeed, since PRM extracts and defines significantly more samples and edges, the resulting paths only have a slightly lesser cost than RRT's. Moreover, RRT is easily adapted and managed, allowing the flexible change of the algorithm to provide a variety of functionalities for motion planning [2] [14] [15] [16].

The principal extension of RRT is its asymptotically optimal version, Rapidly-exploring Random Tree Star (RRT*) proposed in [2]. In addition to RRT, which only connects the samples with the nearest nodes of the tree, RRT* attempts to find an asymptotically optimal trajectory by minimizing the path between the beginning of the tree and the samples. The cost (most generally, the length) of each path is calculated, compared and chosen iteratively as presented in Algorithm 1 [17].

Algorithm 1 Pseudo code for the RRT*

```
G.init(X_root);                                   ▷ initialize tree graph with root node
n_max ← maximum iterations
n_min ← minimum iterations
for n in range(1,n_max) do
    X_rand ← random configuration sample
    X_neighbor ← nearest(V(G),X_rand);            ▷ nearest vertices to X_rand
    X_neighbor ← Obstacle_avoidance(X_rand,X_neighbor )
    X_new ← G.extend(X_neighbor,X_rand);          ▷ add X_rand to the tree graph
    G.rewire(X_neighbor,X_new);                   ▷ rewire tree graph
    if reached_goal(X_new) and n > n_min then
        return find_path()
    end if
end for
return "No path found"
```

The algorithm starts by initializing the root of the tree graph G with the starting position of the robot: X_{root}. We take a sample X_{rand} in the free space and collect the tree nodes $X_{neighbor}$ that are within a defined radius R from this sample. From all of these new branch possibilities, we discard the ones that collide with obstacles, and analyse if the paths can not be optimized in the tree *rewiring* step. Samples will continue being collected until the algorithm has reached the maximum iteration number n_{max}, or reached the minimum iteration n_{min} and found an end-to-end solution. The *Rewire* function is fundamental in RRT* to find the asymptotically optimal trajectory. Its pseudo algorithm is presented in Algorithm 2.

Algorithm 2 Pseudo code for the *Rewire* function in RRT*

```
X_new ← configuration of new node in the tree graph
for neighbor in Neighbors do
    if Cost(X_new) + Distance(X_new, X_neighbor) < Cost(X_neighbor) then
        Cost(X_neighbor) = Cost(X_new) + Distance(X_new,X_neighbor)
        Parent(X_neighbor) = X_new    ▷ add vertex (X_new,X_neighbor) to the tree graph
        Update cost in vertices forward to X_neighbor in the tree graph
    end if
end for
```

The *Rewire* function analyses each of the sample's neighbors $X_{neighbor}$, and analyses whether the cost of passing by X_{new} to reach $X_{neighbor}$ is lower than the current cost

for reaching $X_{neighbor}$. If so, the path for $X_{neighbor}$ is replaced with the new vertex in the RRT* graph. The cost of the nodes following $X_{neighbor}$ are then updated.

The neighbor nodes around X_{new} are the nodes in G within a distance R from X_{new} [18], which is given by

$$R = \gamma \left(\frac{log(|V|)}{|V|} \right)^{\frac{1}{d}}, \tag{2.1}$$

where d is the workspace dimension, γ is the planning constant and $|V|$ is the number of vertices in the tree graph.

As mentioned previously, RRT motion planning algorithm implies less computational burden and time than PRM. It is also able to find trajectories in the free space which can be optimised asymptotically with the extension of RRT, RRT*, by minimizing the costs (eg. distances) of the selected paths.

RRT* Extensions and Applications

Different applications of RRT* extensions have been tested before. In [14], a small modification of RRT* is presented as *informed*-RRT* (i-RRT*). Seeking to minimize trajectory length, a threshold in the form of an ellipse is defined for sample selection. The determination of the ellipse's size is defined by the calculation in Figure 2.1.

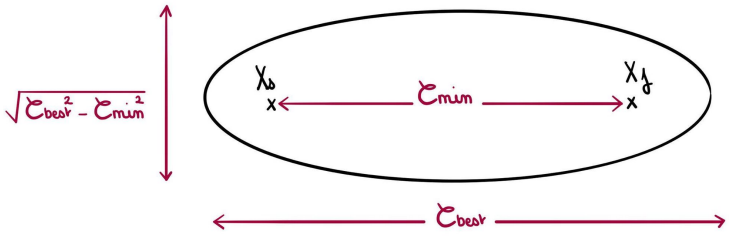


Figure 2.1: Size definition of the elliptical sample domain. Adapted from [14]

This ellipse's shape depends on both the initial (X_s) and goal (X_f) states, the theoretical minimum cost between the two ($C_{\min}$), and the cost of the best trajectory found so far ($Cbest$). If a sample's cost is minor than the best calculated cost so far, the sample with the minor cost is kept and the ellipse's size is reduced. By doing this, we are minimizing the number of samples that can be tested to the most advantageous ones. This method retains the same probabilistic guarantees on finding an optimal solution as RRT* while requiring less computational time and effort.

Meanwhile in Intelligent Bidirectional RRT* (IB-RRT*) [16], the algorithm firstly uses the bidirectional trees (B-RRT*) approach and secondly introduces intelligent

sample insertion for tree connection. B-RRT* builds two independent trees, which are independently constructed and then connected to each other. IB-RRT* returns the nearest vertex on the best selected tree which is eligible to become the connection vertex between both trees, implying a shorter length. [16] explains how this method maintains the asymptotic optimallity property of RRT* and presents a faster convergence although the computational effort is bigger than RRT*.

Another example is the implementation of Transition-RRT* (T-RRT*) in [19]. T-RRT* is a trajectory planning algorithm for robots in disarranged 3D workspaces and in problems also requiring to integrate some additional cost criterion, i.e. transition tests, during the motion planning to influence the trajectory selection. This prevents transitions to unwanted states which will have higher costs and therefore won't be selected.

Finally, and probably the most significant extension of RRT* to this thesis, is its implementation for multi-agent systems, namely Multi-agent RRT*, MA-RRT*. In [15] and [13], trajectories are simultaneously defined for several robots, with a focus in collision avoidance between robots and between robots and the environment.

2.1.2 Multi-agent Motion Planning

Multi-agent systems motion planning started being implemented in the past few decades [20] as a way to extend, and possibly improve, single system's applications by following a "divide and conquer" philosophy. In 2001, [21] explains how multi-agents can improve the system's performance, comparing to single-agents, in terms of both its processing speed and quality of the outcome. Multi-agent systems are mainly used when the tasks to be executed are too long or complicated for just one robot [7]. Problems involving multi-agent motion planning nowadays generally include: multiple task execution [22], monitoring or supervision [20], tracking [23], object detection, exploration [24], etc. Multi-agent motion planning scenarios are mostly related with defining simultaneous trajectories from one end to another while avoiding collisions between robots and between robots and the delimited space. It is then necessary to estimate where each robot will be and will go in each movement to ensure two or more robots are not in the same space at the same instance.

Multi-agent Motion Planning Sampling Algorithms

There are several RRT motion planning approaches to multi-agent systems. The first, and most likely the simplest one, would be to use a graph with vertices and edges to represent the free space. Assuming the robots move in a synchronised way, they reach their corresponding nodes at the same time. In [15], a multi-agent version of Graph

RRT* (G-RRT*) is used to estimate in which edge or vertex each robot should be at a specific time until it reaches its goal. Consequently, it is possible to know when and which edges are free for motion planning, easily avoiding trajectory conflicts between robots. Although it is a very good approach to avoid collisions between robots, this method does not take into consideration the entire space available for the robots to navigate. This results in possibly longer and more limited number of possible trajectories.

As mentioned before, a well-known multi-agent motion planning is the multi-agent RRT* (MA-RRT*). With MA-RRT*, it is possible to obtain optimal trajectories by selecting collision free solutions with minimal length. Indeed, it usually calculates the shortest paths from the root node to every other node in the tree and rewires vertices with new edges which contribute to a path with an inferior cost than the previous one. Since RRT* determines trajectories for each robot, it can easily analyse if two robots are trying to cross the same area in the same instance. To prevent robots from colliding, Yiannis Kantaros and Micheal M. Zavlanos [25] check if the distance of the next configuration of two robots is inferior to the threshold distance required between robots. This is done by calculating the euclidean distance of each robot's future state and comparing with the threshold. If the distance is minor than the threshold, the solution is discarded and the process of finding an alternative trajectory continues. However, if the distance is higher than the threshold, the sample is taken into account and the path is added to the tree if no other eligible sample has a lower cost. It is important to notice there is a risk of collision while moving towards the future states. Even if those are far apart.

2.1.3 Motion Planning with Temporal Logic Specifications

We have seen that motion planning algorithms are used to define collision-free trajectories from a start position to an end goal. However, in some studies, an end-to-end trajectory planning is not enough. We might need the robot to follow other restrictions or preferences during its trajectory. For instance, passing by specified areas, remaining at a determined distance from obstacles, etc. To make this possible, a system of desired properties, defined by Temporal Logic (TL), is applied to motion planning. TL is a user-defined set of formulas with conditions that are used for establishing rules in a robot's trajectory. It applies constraints to the robot's motion behaviors with spatial and/or time restrictions. TL formulas are evaluated and considered as satisfied or violated during a trajectory, modeling motion planning into choosing one that satisfies all or most of them.

The temporal operators for TL formulas are [26]:

- *"Eventually"*: $\mathcal{F}\ \phi \Longleftrightarrow \Diamond\ \phi$, meaning the condition ϕ has to be fulfilled at least

once during the trajectory;

- *"Always"*: $\mathcal{G}\ \phi \iff \Box\,\phi$, meaning the condition ϕ has to be fulfilled every time during the trajectory;
- *"Next"*: $\mathcal{X}\ \phi \iff \bigcirc\phi$, meaning the condition ϕ must be fulfilled after a certain state in the trajectory;
- *"Until"*: $\phi\,\mathcal{U}\,\psi \iff \phi\ U\,\psi$, meaning the condition ϕ must wait before another event ψ is fulfilled in the trajectory;

By combining operators like *"Always"* and *"Eventually"*, new operator modalities are obtained : *"Always Eventually"* or *"Eventually Always"*.

One category of TL is Signal Temporal Logic (STL). It is used in this work to express the user's preferences in the robot's motion behaviour. Similar to TL, STL is a predicate logic based on signals, which provides not only a qualitative metric, but also a quantitative one. In other words, STL identifies whether a signal is violating the user's preferences and quantifies it into Real values. These values are used to increase the corresponding paths' cost and lower their chances of being selected for the final trajectory. If there is no better choice, STL makes it possible for the motion planning to select the trajectory that *least* violates the preferences.

The predicate for STL is defined as $\pi ::= \mu(s,t) \geq 0$, where μ is a real-valued function of the states, and $\pi : \mathbb{R} \to \mathbb{B}$. Given a state s at a time t, a predicate π is evaluated as true if $\mu(s,t) \geq 0$, and false if $\mu(s,t) < 0$. We consider STL formulas ϕ that are defined recursively as follows:

Definition 1 (STL grammar) [27]:
The basic grammar of STL is defined as follows:

$$\phi ::= True \mid \pi \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2, \tag{2.2}$$

Where True is the Boolean true, π a predicate, $\neg$ the Boolean negation, $\wedge$ the conjunction operator, and $\vee$ the disjunction operator.

Definition 2 (STL Semantics [27] [28] [29]): The validity for an STL formula ϕ with respect to signal s at time t_i is defined as:

- $(s,t_i) \models \top$, if true
- $(s,t_i) \models \pi$, if and only if $\mu(s,t_i) \geq 0$,

- $(s, t_i) \models \neg\phi$, if and only if $\neg((s, t_i) \models \phi)$,
- $(s, t_i) \models \phi_1 \vee \phi_2$, if and only if $(s, t_i) \models \phi_1$ or $(s, t_i) \models \phi_2$.
- $(s, t_i) \models \phi_1 \wedge \phi_2$, if and only if $(s, t_i) \models \phi_1$ and $(s, t_i) \models \phi_2$

  And with temporal operators:
- $(s, t_i) \models \Diamond_{[a,b]}\phi$ if and only if $\exists t'_i \in [t_i + a, t_i + b], (s, t'_i) \models \phi$,
- $(s, t_i) \models \Box_{[a,b]}\phi$ if and only if $\forall t_i \in [t_i + a, t_i + b], (s, t'_i) \models \phi$.

A signal s satisfies ϕ, denoted by $s \models \phi$, if $(s, t_0) \models \phi$.
$(s, t_0) \models \Box_{[a,b]}\phi$ if ϕ holds at every time step between a and b, and $(s, t_0) \models \Diamond_{[a,b]}\phi$ if ϕ holds at some time step between a and b. Indeed, a STL formula ϕ is bounded-time if it contains no unbounded operators, i.e. no operators with "[a, b]". If ϕ is bounded-time, the satisfaction of ϕ of future predicted signals s needs to be calculated during the sum of the designated bounds lengths, otherwise it continues indefinitely.

Robustness Satisfaction Signal

As mentioned before, STL provides information on by *how much* the formula is true or false through the use of its *quantitative semantics*. This quantitative values allows us to estimate the robustness of STL formulas. To calculate robustness we consider the worst-case, meaning only the worst violation of STL specifications will be taken into account. The robustness of STL formulas is provided by a real-valued function ρ of signal s and time t.
The quantitative semantics of all operators and formulas with respect to signal s is defined inductively as follows.

Definition 3 (Robustness Metrics [27] [28] [29]):
Where $\rho_{max} \gg 0$:

- $\rho(s, t, \top) = \rho_{max}$
- $\rho(s, t, \pi) = \mu(s, t)$
- $\rho(s, t, \neg\phi) = \neg\rho(s, t, \phi)$
- $\rho(s, t, \phi_1 \vee \phi_2) = \max(\rho(s, t, \phi_1), \rho(s, t, \phi_2))$
- $\rho(s, t, \phi_1 \wedge \phi_2) = \min(\rho(s, t, \phi_1), \rho(s, t, \phi_2))$

- $\rho(s,t,\Diamond_{[a,b]}\phi) = \max_{t' \in [t+a,t+b]} \rho(s,t',\phi)$
- $\rho(s,t,\Box_{[a,b]}\phi) = \min_{t' \in [t+a,t+b]} \rho(s,t',\phi)$

Theorem 1: [27] [29] For any $s \in S$ and STL formula ϕ, if $\rho(s,t,\phi) < 0$ then s violated ϕ at time t, and if $\rho(s,t,\phi) > 0$ then s satisfies ϕ at time t. This is:

$$\begin{aligned} \rho(s,t,\phi) < 0 &\rightarrow (s,t) \not\models \phi, \\ \rho(s,t,\phi) > 0 &\rightarrow (s,t) \models \phi. \end{aligned} \tag{2.3}$$

The robustness is the link between the STL formulas and their influence on motion planning. A robustness value is attributed to a signal and it represents with real values how much this signal is violating or respecting the user's preferences.

2.2 Related Work

2.2.1 Motion Planning with Temporal Logic

STL constraints are usually used in motion planning to model trajectories according to user's preferences or to features in the environment. These constrains can be both temporal and/or spatial. The paper in [5] focuses on aerial exploration of unknown spaces and uses STL formulas to define constraints in the motion planning. The trajectories are indicated to follow certain user-defined spatial preferences defined by STL formulas, such as minimal and maximum distances of the robot's position with the obstacles in its environment. Applying these preferences to the robot's motion behaviour, increases the predictability of its future position. Due to its minimal distance constraints, it can also be used to define safe trajectories by remaining at a safe distance from obstacles. This work allows the exploration and mapping of unknown spaces, including narrow or dangerous spaces which are difficult for humans to explore. Besides focusing on the quantity of information extracted and time needed to explore, [5] also considers the structure of the exploration by analysing the robot's strategic positions to important features in the environment to enhance more detailed and robust explorations. The path selection is made with the comparison of the costs of each eligible paths defined by RRT* and respecting the STL preferences. Each path segment is associated to a gain weighted with a cost function that takes in consideration the quantitative satisfaction of STL preferences. The segment with the lesser gain is then selected to take place in the RRT* graph. In [30] (2017), Vasile, Raman and Karaman propose a sampling-based algorithm RRT* with STL specifications. In this algorithm they use a class of

heuristic functions based on the *Direction of Increasing Satisfaction* (DIS). The DIS allows the collection the most promising samples to improve the satisfaction of STL formulas.

Another used category of Temporal Logic is Linear Temporal logic (LTL). In contrast with STL, LTL is a set of formulas which are discretely evaluated during motion planning. With LTL, formulas are either respected or not: their evaluation returns Boolean answers instead of quantified values like in STL. In a recent paper [31], LTL formulas are used to define the traffic rules of the road. In fact, LTL can be used to ensure the robot safety and the safety of others around it in a motion planning algorithm of a vehicle circulating in a road network. The formulas can specify desired behaviours related to the speed limits, the presence of construction zones or traffic lights, etc. With LTL, there is no information about the quantitative satisfaction of these constraints during motion planning, therefore it randomly chooses one of the satisfying trajectories.

2.2.2 Multi-agent MP with Temporal Logic

Temporal logic can also be applied to multi-agent motion planning. Indeed, TL allows the distribution of tasks between robots by assigning different formulas to different robots. This way, each robot has its own spatial and temporal preferences to follow. It is possible to observe that there are not many works based on multi-agent motion planning using STL. One highlighted work from 2017 [32] on multi-agent motion planning, uses STL to avoid collision between agents or with possible objects, while maintaining an optimized communication quality of service between agents. STL specifications are firstly used to ensure robots keep a minimum distance between each other and obstacles so they will not collide. And secondly, they are used to control in which and by how many time steps each robot should be in a specified area in order to optimize the communication conditions.

Nonetheless, despite not being able to find many works involving multi-agent motion planning using STL, it is possible to find numerous works using LTL instead. In recent papers [33] [25], Yiannis Kantaros and Micheal M. Zavlanos propose a new optimal control synthesis algorithm for multi-agent systems where LTL specifications are used to establish rules for each robot's trajectory. The LTL formulas are represented by a Non-deterministic Büchi Automaton (NBA) which, along with the transition systems, defines the state-space of the product automaton. The algorithm then incrementally builds directed trees to represent the state-space and the transitions between states of the product automaton. In order to get the optimal trajectories, RRT* is used to find the best solution to reach a goal in the tree by finding the minimal path cost. Another

work done in multi agent motion planning with temporal logic constraints is the automatic path planning for a team of robots in [34]. The team's mission specifications are expressed with LTL, using an optimizing proposition that must be repeatedly satisfied. If one robot is not able to complete its exact trajectory, a communication with other robots is established to ensure the LTL specifications are not breached.

Chapter 3

Methods

The different steps of the desired motion planning algorithm for multi-agent systems is presented in Figure 3.1 where STL specifications functions are included to represent the influence they have on this motion planning algorithm.

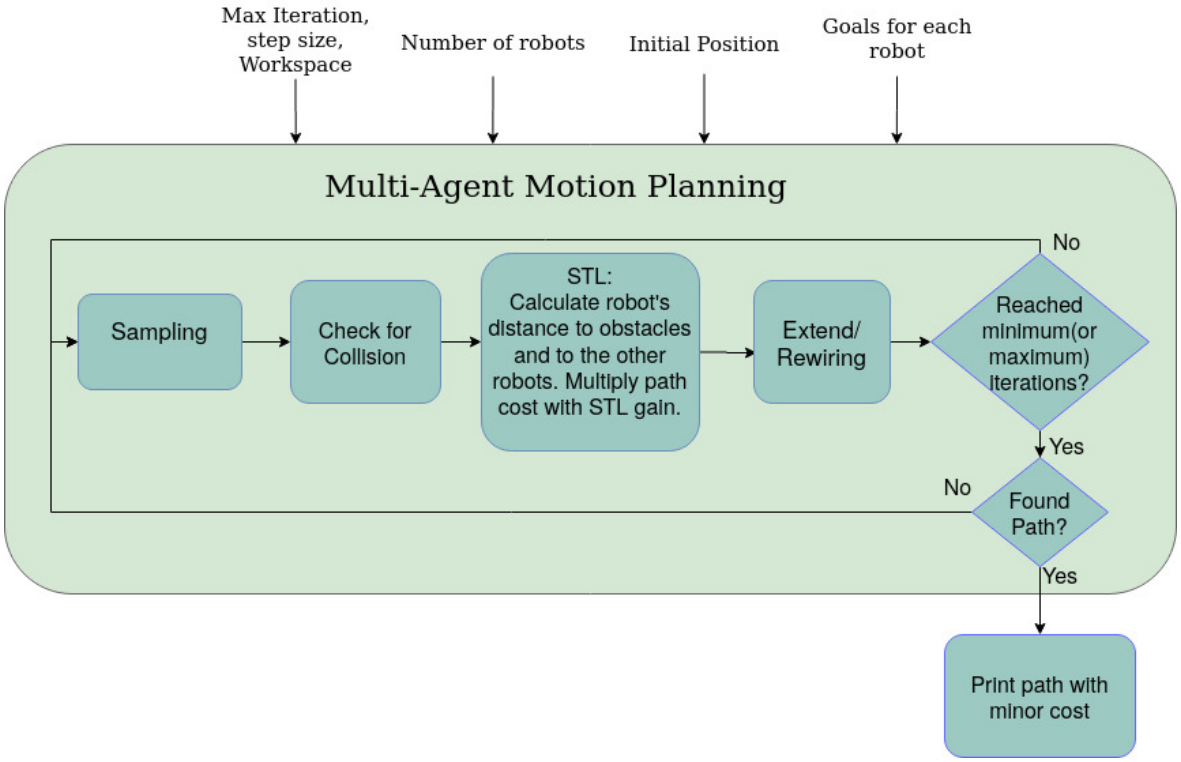


Figure 3.1: Scheme of the proposed multi-agent motion planning using STL constraints

In this Chapter we will have the chance to study in more details the thesis approach of this motion planning algorithm.

3.1 Temporal Logics

As seen in Figure 3.1, Temporal Logic (TL) is applied to this project in the shape of Signal Temporal Logic. As explained previously, STL formulas are responsible for specifying preferences to be followed by the robot's movements and behaviours while it reaches for its goals.

3.1.1 Signal Temporal Logic

Let us consider a team of n agents conducting motion behavior in the shared environment X. In this thesis work, we desire to establish minimum and maximum distances between robots and between robots and the environment.To simplify this study, we consider all agents share a synchronized clock and that each movement of the robots to their future states takes more or less the same amount as other robots'. We assume there is one common STL formula for every robot.

In this work only the **Global** operator $\mathcal{G}$ will be taken into consideration for the STL formulas. In other words, the robots should always respect the preferences imposed by the STL. Furthermore, the formulas will mainly include minimum and maximum distances to robots and to static obstacles in the environment. As formulated in [5], the STL formula responsible for verifying if the robots are respecting the distances required from the different obstacles in the environment is given by:

$$\begin{aligned} \phi_{obstacles} &= \bigwedge_{\substack{i=[1,n] \\ k=[1,\#obs]}} \text{dist}(r_i, obs_k) \geq D_{min,obs} \\ \theta_{obstacles} &= \mathcal{G}(\phi_{obstacles}) \end{aligned} \tag{3.1}$$

Where $\#obs$ is the number of obstacles, n is the number of robots and 'dist(a,b)' is the euclidean distance between points a and b in the 2D environment X. This formula ensures robot r_i will avoid collisions with obstacles obs_k in the environment by maintaining a distance of at least $D_{min,obs}$ from them.

The STL formula responsible for verifying if the robots are respecting the distances required from the remaining robots in the environment is given by:

$$\begin{aligned} \phi_{robots} &= \bigwedge_{\substack{i=[1,n-1] \\ j=[2,n] \\ i<j}} \text{dist}(r_i, r_j) \geq D_{min,robot} \\ \theta_{robots} &= \mathcal{G}(\phi_{robots}) \end{aligned} \tag{3.2}$$

This formula ensures robot i will avoid collisions with other robots j, where $i \neq j$, by maintaining a distance of at least $D_{min,robot}$ from each other. The formula θ_{robots} is very restrictive and if a maximum distance preference is included in the formula, θ_{robots} might become infeasible for $n > 3$. It is also possible to modify the first formula and use it for defining a formation, i.e. just a conjunction of the desired distances between some of the agents. Furthermore, by using OR operator $\vee$, it could be possible to order a robot $robot_1$ to stay close to $robot_2$ or $robot_3$ for example.

Finally, motion planning preferred behaviours for all agents in this work are summarized in the following STL formulas:

$$\begin{aligned} \phi_{final} &= \phi_{obstacles} \wedge \phi_{robots} \\ \theta_{final} &= \mathcal{G}(\phi_{final}). \end{aligned} \tag{3.3}$$

In formula (3.3), we state that both the minimum distances from obstacles and the minimum distance from robots has to be respected.

Robustness Satisfaction Signal

As mentioned before, only the worst-case violation of STL formulas is considered for robustness. In equation (3.2), this is done by searching for the minimum distance between robot i and the other robots in its surroundings. For each robot r, the minimum distance to other robots r_j is given by:

$$\phi_{robots} = \min_{\substack{i=[1,n-1] \\ j=[2,n] \\ i<j}} (\text{dist}(r_i, r_j) - D_{min,robot}). \tag{3.4}$$

Where $D_{min,robot}$ corresponds to the preferred distance set by the STL formula. In equation (3.1), this is done by searching for the minimum distance between robots i and the obstacles in its surroundings. For each robot r, the minimum distance to obstacles obs is given by:

$$\phi_{obstacles} = \min_{\substack{i=[1,n] \\ k=[1,\#obs]}} (\text{dist}(r_i, obs_k) - D_{min,obs}). \tag{3.5}$$

Where $D_{min,obs}$ corresponds to the preferred distance set by the STL formula.

The robustness degree of a formula $\theta = \mathcal{G}(\phi)$ along a path $s(\alpha)$, $\alpha \in [0,1]$ (or, as showed before, we can also use s at time instant t) is calculated as:

$$\rho(\theta, s) = \min_{\alpha=[0,1]} \rho(\phi, s, \alpha). \tag{3.6}$$

When we are considering both STL formulas as in equation (3.3), we need to determine which of both is violating the STL specifications the most. Considering the signal s is the path being studied, we define for each of the different formulas above, the ρ function recursively as:

For equation (3.2), the robustness is calculated as:

$$\begin{aligned} \rho(\theta_{robots}, s) &= \min_{\alpha\in[0,1]} \rho(\phi_{robots}, s, \alpha) \\ &= \min_{\alpha\in[0,1]} \Big(\min_{\substack{i=[1,n-1] \\ j=[2,n] \\ i<j}} (\text{dist}(r_i, r_j) - D_{min,robot}) \Big) \\ &= \min_{\alpha\in[0,1]} (\min(d_{robots})). \end{aligned} \tag{3.7}$$

For the equation (3.1), the robustness is calculated as:

$$\begin{aligned} \rho(\theta_{obs}, s) &= \min_{\alpha\in[0,1]} \rho(\phi_{obs}, s, \alpha) \\ &= \min_{\alpha\in[0,1]} \Big(\min_{\substack{i=[1,n] \\ k=[1,\#obs]}} (\text{dist}(r_i, obs_k) - D_{min,obs}) \Big) \\ &= \min_{\alpha\in[0,1]} (\min(d_{obstacles})). \end{aligned} \tag{3.8}$$

Finally, for the equation (3.3), the robustness is calculated as:

$$\begin{aligned} \rho(\theta_{final}, s) = \rho(G\phi_{final}, s) &= \min_{\alpha=[0,1]} \rho(\phi_{final}, s, \alpha) = \min_{\alpha=[0,1]} \rho(\phi_{robots} \wedge \phi_{obstacles}, s, \alpha) \\ &= \min_{\alpha=[0,1]} \Big[\min_{k=[1,\#obs]} (\text{dist}(r, obs_k) - D_{min,obs}), \\ &\min_{\substack{i=[1,n-1] \\ j=[2,n] \\ i<j}} (\text{dist}(r_i, r_j) - D_{min,robot}) \Big] \\ &= \min_{\alpha=[0,1]} [\min(d_{robots}, d_{obstacles})]. \end{aligned} \tag{3.9}$$

The respective algorithm for STL robustness calculation is presented in Algorithm 3.

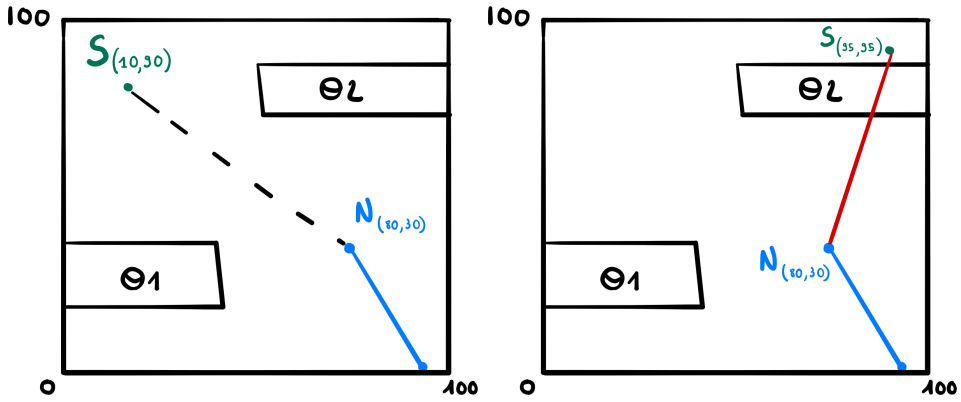
Algorithm 3 Pseudo code for STL robustness calculation

1: $X_{rand} \leftarrow$ sample configuration
2: $X_{neighbor} \leftarrow$ configuration of neighbor
3: $distObs \leftarrow$ minimum distance of edge to obstacles
4: $distRobs \leftarrow$ minimum distance of edge to other robots
5: $dist \leftarrow$ min(distObs,distRobs)
6: STL_$robustness \leftarrow$ calculus of equation (3.8)
7: **return** STL_$robustness$

3.2 Multi-agent RRT*

Previously we have seen multi-agent motion planning can be derived from single-agent algorithms. In this work, MA-RRT* motion planning algorithm is used to determine non-conflicting trajectories for a defined number of robots. In the work [15] mentioned above, only edges or vertices of the free space graph are considered when defining new configurations in robots' trajectories, i.e., if a robot traverses a vertex V in moment t to reach a certain edge E, both V and E will be unavailable to traverse for other robots. Meanwhile in this work, all free space plan will be considered for trajectory planning instead of having a grid workspace. This leads to having more options in trajectory selection, making it easier to single out trajectories which avoid collisions between robots.

There are several steps to turn a RRT* into a Multi-agent RRT*. Firstly, the samples' dimension need to be adapted according to the number of robots involved. For the following description we consider a 2D workspace. In case there is only one robot involved, the samples will be in the form: $S = (x_1, y_1)$. Since there is only one robot, there is no need to worry about inter-robot collisions. Obstacle collision is avoided by checking if there is any intersection between the added edge, which consists in the link of new sample with the chosen tree node, and the obstacles in the workspace. Having one robot, the edge only includes a single link.
Figure 3.2 illustrates the collision avoidance algorithm explained above.

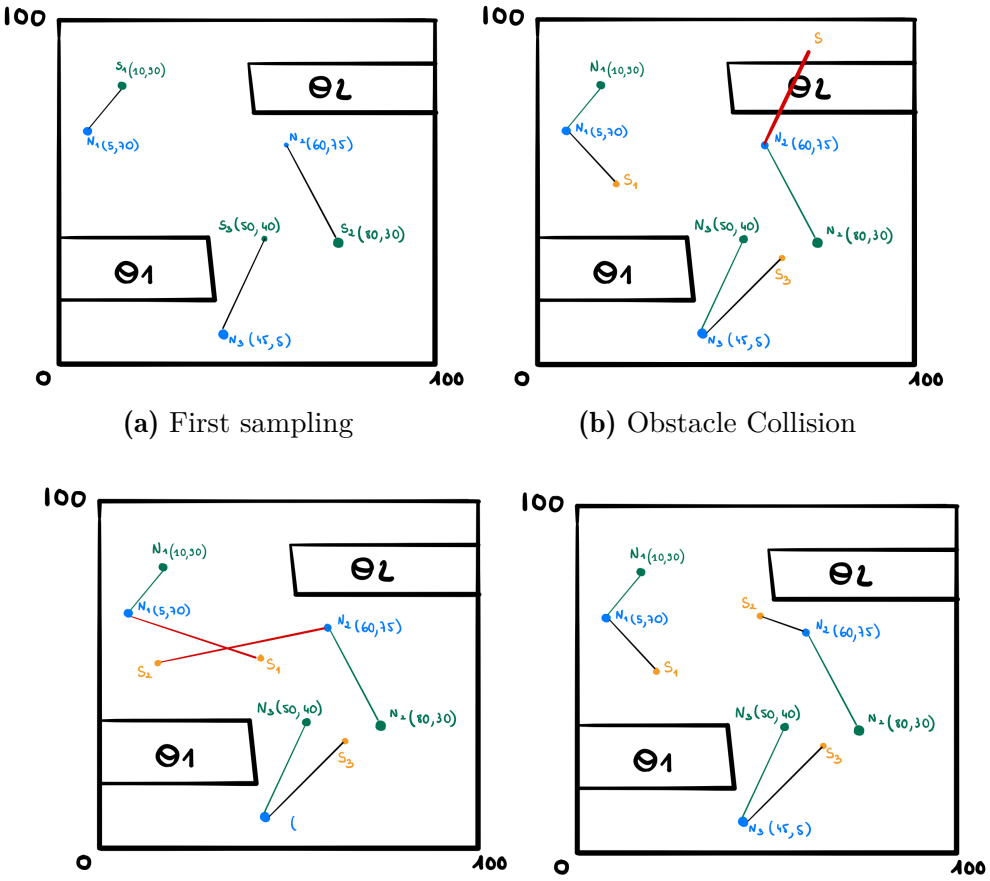


(a) Without collision **(b)** With collision

Figure 3.2: One robot sampling process with collision avoidance

When selecting a sample in the free space, we consider new vertices that connect the sample to neighbor tree nodes. The collision avoidance algorithm analyses the minimum distance of these vertices to obstacles, if the distance of the new vertex (black dots in Figure 3.2a) is superior than zero the sample is added to the tree. If the distance of the new vertex (red line in Figure 3.2b) is equal to zero the sample is discarded.

In case there is more than one robot, the samples will be in the shape $S = ((x_1, y_1), (x_2, y_2), ..., (x_n, y_n))$, where n corresponds to the number of robots. The size of S is therefore: $len(S) = \#_{Robots} \times Dim_{workspace}$. As observed, the samples collected in MA-RRT* are composed by as many individual configurations as the number of robots. Indeed, each single sample s inside the sample S is associated to one robot. When checking for collisions with obstacles, the algorithm verifies if there is no intersection between obstacles and the edges connecting each individual sample s in the collective sample S with the respective single configurations in the neighboring node. To check for collision between robots, the algorithm compares each of these links with one-another and verifies there is no intersection between them. Figures in 3.3 illustrate the description for collision checking made above.



(a) First sampling **(b)** Obstacle Collision

(c) Inter-Robot Collision. **(d)** Second Sampling

Figure 3.3: Three robot sampling process with collision avoidance.

In figure 3.3a we observe there are three robots in the free space with position $N = (N_1, N_2, N_3) = ((5, 70), (60, 75), (15, 5))$. A successful sample S without collisions was already collected. In figure 3.3b, the previous sample now belongs to the tree as N. A new sample S is taken. We notice the sub-sample S_2 can not be reached since the (red) path that connects it to the tree collides with obstacle O_2. This sample is then discarded and another sample is collected. In Figure 3.3c, the new sample does not collide with any obstacle. Assuming we know the robots' velocities, we know where each robots will stand at each instance when traversing a path. By comparing where each robot is along the new path we can discard the sample if they come to close to each other. This is the case for the paths connecting N_1 to S_1 and N_2 to S_2. Robots 1 and 2 are very likely to collide with each other since their trajectories are intersected in a critical place. This sample is then discarded and a new sample is collected, Figure 3.3d. This time, we observe there are no chances of collision either with obstacle or between robots. This sample is added to the tree.

Collision avoidance is done by excluding any sample located inside an obstacle, by excluding edges that intersect with obstacles or with other robots' trajectories. In [25], the collision avoidance between robots is done by setting a minimum distance between the robots' vertices configurations. This thesis work's algorithm proposes a collision avoidance between edges of the tree instead. This allows the algorithm to verify robots do not collide with each other while travelling the edges of the tree to reach a certain vertex configuration in the free space. By analysing travelled edges, it possible to verify if the entire trajectory is safe from collisions. In addition, the collision avoidance is also reinforced by the STL formulas.

STL formulas define the desired minimum and maximum distances of robots from obstacles and from other robots. These distances are found by calculating the distances to objects along an edge of the Tree. The minimum and maximum distances between robots are found by calculating the distance between robots along their respective individual edges connecting two nodes. When collisions are detected, they can be avoided either through the Multi-agent RRT* by discarding samples inside of obstacles [13] and samples that cause collisions between robots, or as seen in [35] through an increase in path costs when they do not respect the STL formulas.

3.2.1 Path Cost and Trajectory Planning

The advantage of using RRT* is that it searches for the path with the smallest length and cost between final positions. In this work, the cost related to each edge will be influenced by the euclidean distance between the two vertices (length of the corresponding path edge) and by the cost generated with the STL robustness. The preliminary cost of an edge is given by the euclidean distance between both of its vertices.

Cost calculation for an edge s_k^{k+1} between two vertices v_k and v_{k+1}:

$$C(s_k^{k+1}) = ||v_k - v_{k+1}||^2 \tag{3.10}$$

When several robots are involved, the global cost is the sum of the individual costs for each robot, as seen in equation (3.11).

Total cost calculation for an edge s_k^{k+1} between two vertices v_k and v_{k+1} for all robots:

$$C(s_k^{k+1}) = \sum_{i=1}^{\#Robots} C(s_{i_k}^{k+1}) \tag{3.11}$$

To influence the motion planning into selecting a path which better respects STL specifications, we introduce the STL formula's robustness. The robustness is used as in [5] and is reflected by the cost given in equation (3.12).

Value reflecting STL formula's robustness for the edge s_k^{k+1}:

$$Robustness_{cost} = max(e^{-\beta\rho(\theta,s_k^{k+1})}, 1) \tag{3.12}$$

Where β is a tuning parameter which defines the importance of respecting STL formulas. The higher the β, the higher the cost is if STL preferences are being violated. The higher the cost, the less likely it is for a violating path to be selected in the motion planning. If β is low, STL constraints will have less influence on the motion planning. It is important to find a balance between respecting STL formulas and being able to slightly break these preferences if it shows to be advantageous for the global trajectory. For instance, motion planning can decide to chose a shorter path which is slightly violating the rules instead of taking a path with a big detour for which the cost is higher.

Lastly, the final cost of the edge s_k^{k+1} is given by multiplying the euclidean distance between the two vertices calculated in (3.11) with the corresponding $Robustness_{cost}$ obtained with (3.12). We can deduce the final cost of an edge s_k^{k+1} between two vertices v_k and v_{k+1} is given by:

$$C(s_k^{k+1}, \theta) = C(s_k^{k+1}) \times max(e^{-\beta\rho(\theta,s_k^{k+1})}, 1) \tag{3.13}$$

Equation (3.12) implies that the robustness will only influence the edge cost if the STL formula is being violated. If there is no violation, the $Robustness_{cost} = 1$ and only the euclidean distance will be taken into account for the final cost. Indeed if $\rho(\theta, s_k^{k+1}) < 0$, then $e^{-\beta\rho(\theta,s_k^{k+1})} > 1$.

Tree Construction and Rewiring

The cost of each vertex in the tree corresponds to the cost it takes for the robot to travel from the starting point to that vertex. In other words, it is the sum of the travelled edges' costs to reach that vertex. As seen in [5], the cost to reach a vertex V is given by:

$$C(V) = C(pa(V)) + C(s_{pa(V)}^{V}, \theta) \tag{3.14}$$

Where pa(V) is the parent Vertex of V.

Each time the tree is extended by a new vertex V, the RRT* elaborates a search to look for alternative paths within the tree that would imply lower path costs. The vertices are rewired if their cost can be minimized. The cost of the forward vertices are then reduced by the cost difference between the new and the old path.

3.2.2 Trajectory Planning Approach

The main objectives of this centralized multi-agent motion planning algorithm are (1) to define collision free trajectories, (2) to find asymptotically optimal solutions and (3) to have a low computational time. Objective (1) is ensured with the RRT* collision avoidance algorithm and by selecting paths that respect STL constraints. Objective (2) is achieved by finding paths with the smallest costs and smallest distances while respecting STL constraints. Finally, objective (3) is obtained by reducing unnecessarily long tasks. For instance, by using biased sampling 3.2.2 and by using the K-nearest neighbors 3.2.2 strategy.

Steps of the proposed solution

The diagram of the developed solution for this work is represented in Figure 3.4. and its pseudo-code is presented in Algorithm 4.

Algorithm 4 Pseudo code for the solution Ma-RRT*

```
#_robots,X_start, Goals_list ← parameters regarding robots
max_goals ← maximum number of goals the robots have to reach
n_max, n_min ← max and min iterations
X ← workspace
for i in range of max_goals do
    Goals ← [ ]
    for rob in range of robots do
        # Next goals to be reached by each robot
        Goals.append(Goals_list[rob][i])
    end for
    # Initialize the Tree graph
    rrt = rrtStar(X, #_robots, X_start, Goals, n_max, n_min, etc.)
    # Increment the trajectory as the goals are reached in the Tree graph
    final_trajectory += rrt.BuildTree()
end for
return final_trajectory
```

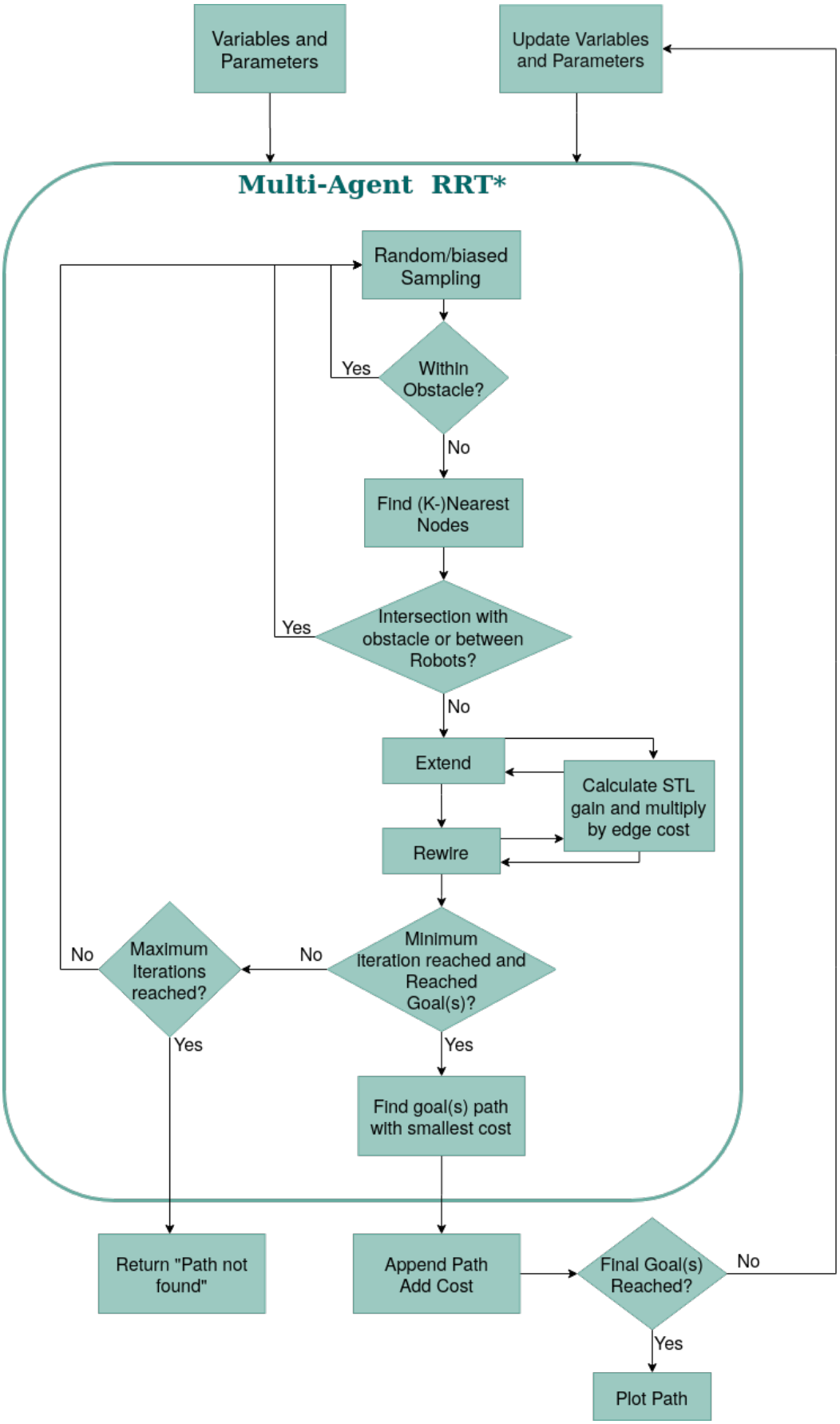


Figure 3.4: Diagram of the developed multi-agent motion planning.

Variables and Parameters

The program starts by initializing the necessary variables and parameters. For instance, the number of robots, their start configuration and their desired goal area. The maximum step size per edge, the maximum and minimum number of iterations. The workspace environment, the number K of maximum nearest neighbors and the STL tuning parameter β. With this information, the RRT graph is initialized and the ma-RRT* with STL constraints can start running. If the robots have are more than one goal, the algorithm will deal with one goal per robot at a time and concatenate the trajectories in the end, see Algorithm 4.

Random and Biased Sampling

In this algorithm we intend to have the robots reach their goals at the same time. However, the higher the number of robots, the harder it is to get random samples inside of the desired goal areas. This results in very long computational time. The program runs until the trajectories are completed and therefore until these specific samples appear. A way to decrease computational time is to manually provoke this type of samples. In this work, about 25% of all samples are biased in order to be located in the robot's desired goal areas. The rest of the samples collected are random. This fastens the search for end to end trajectories for all robots. Using an existent path to bias the sample as in [14] is not possible for this thesis work since the edges have different costs because of STL's robustness effect on the cost function in equation (3.13).

K-nearest Nodes

For each new sample, the algorithm collects all Tree nodes that are within a radius R from the sample. It studies each neighbor to analyse which is the best place for the sample in the tree. When the tree starts having a considerable size, the number of neighbors increases and the computational time is higher. To reduce the computational time, the maximum number of neighbor nodes collected is set to K.

Collision Avoidance

RRT* collision avoidance verifies the edges connecting neighbors to the new sample are not intersecting with obstacles nor with other robots' edges. It returns the neighbors that do not provoke collisions.

Extend and Rewire with STL

The *extend* function calculates the cost (euclidean distance × STL robustness value) of the edges connecting the remaining neighbors to the sample and adds the one with lesser cost to the Tree graph. The *rewire* function checks whether it is possible to reduce the costs of the neighbor nodes by passing through the newly added edge. If so, the Tree is rewired and the cost of the forward nodes is updated.

Reached Goal(s)

Once the program has reached the minimum iteration number, the program checks if the added node is located in the robots' desired goal areas. If not, the program returns to the sampling process until a sample positioned inside the goal areas is added to the Tree. If so, the program returns the trajectory with the smallest cost, its respective cost. This trajectory is then added to the global trajectory if there are more goals to be reached.
If the maximum number of iterations is reached before finding a feasible trajectory, the program returns that no solution was found.

Update and Program Completion

If there are any other goals to be reached, the current position and new goals of the robots are updated. The MA-RRT* re-runs with the updated parameters. This is done until no more goals have to be reached. The trajectories are then saved and plotted, alongside with the cost and computational time.

This algorithm delivers asymptotically optimal solutions. In theory, it should deliver the trajectories with shorter size and that respect the STL constraints. The more iterations we run, the more samples we collect and the more likely it is to find trajectories with smaller costs. Moreover, STL preferences should only be violated when necessary or when no alternative trajectories were found.

Chapter 4

Simulations and Discussion

In this section, simulations testing several core aspects of the project are presented and the results are compared and analysed. The proposed solution is tested through simulations and experiments by applying user's preferences with STL formulas and examining the resulting trajectories. The code is written in python and the CPU of the computer running the algorithm is Intel® Core™ i7-6500U CPU @ 2.50GHz x 4.

4.1 Test Workspace

The example of a common workspace used to represent the environment is presented in Figure 4.1. The size of the 2D workspace is 100×100. It contains seven obstacles, o_1 to o_5, represented by grey rectangles. There are five rectangular regions represented in green, which consist in the five possible goal areas.

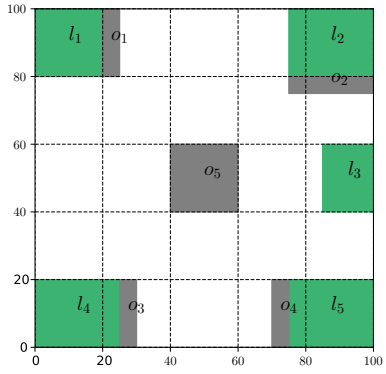


Figure 4.1: 2D workspace used for simulations.

4.2 Parameters for simulation

The parameters to be considered for this solution are the following:

Ma-RRT*:

- number of robots;
- starting position and goal areas;
- n: number of minimum iterations;
- step size: maximum distance between samples and the Tree;
- k: maximum number of nearest neighbors,

STL:

- D_{obs}: parameterized distance from obstacles;
- D_{robots}: parameterized distance from robots;
- β: STL tuning parameter;

4.3 Multi-agent RRT*

In this solution, the Multi-agent RRT* algorithm finds a set of trajectories in which all robots reach their goals at the same time. This means each robot will follow a path with the same number of intermediate configuration as other robots. This makes it easier to prevent collisions and to obtain preliminary trajectories respecting the user's preferences. Further in the results, an increment stage to this approach is done which will optimize each robot's trajectories. In theory, this solution is a collision-free trajectory which begins in the robots' starting point, passes by their intermediate goals (if any), and ends in their final goal.

4.3.1 Effects of Adding Biased Sampling and Limiting the Number of Neighbors to a Maximum of k-Nearest Neighbors in MA-RRT*

We start by running the developed motion planning to define trajectories for three robots without using biased sampling and without a limit of k-nearest neighbors. We

compare the performances of the MA-RRT* in its simplest version with the MA-RRT* using biased sampling (BS) and the MA-RRT* using BS and with k-nearest neighbors.

The parameters for this simulation are in the caption of Figure 4.2. The minimum iteration number n is set to 2.000. This means that from 2.000 iterations, the algorithm can stop if at least one solution is found after that. If a solution is found, the algorithm returns it and saves it. However, if a solution is not found, the algorithm can run up until 50.000 iterations to find one. If it reaches this number and still no solution is found, the algorithm returns that it did not find a solution. The step size is set to 60 to allow bigger steps to be taken in the beginning of the Tree construction. During the first iterations, the Tree is still small and the samples will most likely be located away from the Tree. A bigger step size in the start, allows the algorithm to add these first samples to the Tree. As the Tree begins to expand, smaller step sizes are needed and the radius delimiting the area of the nearest neighbours, in equation (2.1), is reduced. Finally, after comparing the influence the different values β have in the motion planning (Appendix section A.2), we concluded that $\beta = 4$ delivers a good performance and makes sure users' preferences are respected.

In Figures 4.2 and Appendix A.1, we can observe the trajectories obtained for the different versions of the MA-RRT*. Robot $R0$ starts from position (0.0, 0.0) in the workspace and travels to green area $l3$. Robot $R1$ starts from position (100.0, 0.0) in the workspace and travels to green area $l2$. And finally, robot $R2$ starts from position (0.0, 100.0) in the workspace and travels to green area $l4$. All of the starting positions are represented by zeros in the workspace and the order of states travelled is indicated. Table 4.1 presents their respective performances. It is possible to notice that the trajectories obtained by using BS and a 150-nearest neighbors limit (Figure 4.2c) are simpler than than with the other two versions. In fact, in Table 4.1, we observe the MA-RRT* with BS and a limit of 150-nearest neighbors has the lowest cost of the three versions. Moreover in Figures 4.2 we can see that the simplest version of MA-RRT* has a trajectory of 9 intermediate configurations states when the other two versions have only 6. In other words, in the first version robots will need to travel through 9 different configurations until they reach their final goals. Meanwhile in the other two versions, robots only need to travel through 6 configurations.

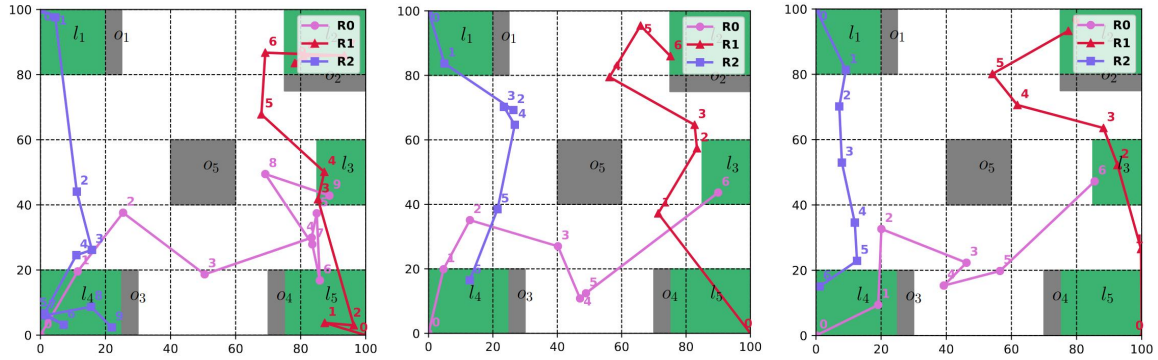


Figure 4.2: Three robots' trajectories with (a) simple, (b) with biased sampling (BS), (c) with BS and *150*-nearest neighbors MA-RRT*. The initial positions are marked with zeros and the order of states travelled is indicated. With parameters: $n = 2.000$, step size=60, $\beta = 4$, $D_{min,robots} = 10$, $D_{min,O1-4} = 5$ and $D_{min,O5} = 10$.

Table 4.1: Performance of simulations in Figures 4.2 and A.1.

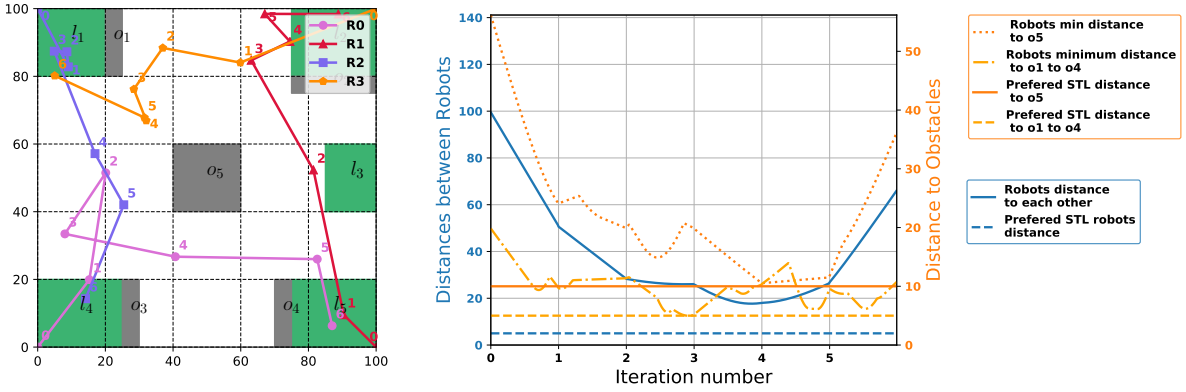
	MA-RRT*	"" w/ Biased Sampling(BS)	"" w/ BS and K-nearest neighbors
Iteration number	4.355	2.023	2.015
Total_time	16m11s89ms	5m56s22ms	5m11s54ms
Path cost	75489.09	380.72	365.80

In the Figure of Appendix A.1 we see that none of the STL formulas for inter-robots distance are breached. However in sub-Figure of Appendix A.1a, we note that the STL preferences of distance to obstacles are violated in the end of the trajectory. Robots $R1$ and $R2$ breach the preferred 5 meters minimum distance to obstacle 6. This translates into a higher path cost, as seen in Table 4.1. The algorithm did not find a solution which did not violate the STL formulas. As mentioned before, in this first version it is hard to find a sample containing all of robots' goals. When a sample is collected within these areas, it has high probabilities of being selected in the final trajectory since there are not many of these to chose from. The wait for samples like these to be collected is the main reason for the high computational time implied in this version. It runs up until 4355 iterations to finally find a solution. The computational time is considerably reduced from the simplest version to the version using BS. This shows the utility of manually introducing samples within the green areas. If they are collected more frequently, it is easier and faster for the algorithm to find solutions which better respect STL formulas. For instance, in the versions of Figures in Appendix A.1b and

A.1c, the STL formulas concerning the distance to obstacles is respected. Using the *k-nearest* neighbors also allows to reduce computational time as is reduces the number of nearest nodes to be evaluated when adding a new sample to the Tree.

4.3.2 Four Robots Trajectory - Scalability of the Solution

To test the scalability of the solution, we increase the number of robots to four. The parameters used for this simulation are presented in the caption of Figure 4.3. To obtain good results, it is necessary to run the program with a much higher number of iterations than when using the program for three robots. In fact, it is harder to synchronize the trajectories of four robots, they need to be free of collisions and with the smallest possible cost. For this reason, in the following simulation we used a minimum of 20.000 iterations. The simulation trajectories are illustrated in Figure 4.3b, and its performance is presented in Table 4.2.



(a) Robots' simulated trajectories. **(b)** Robots' distances to each other and to obstacles.

Figure 4.3: Four robots' trajectories where the initial positions are marked with zeros and the order of states travelled is indicated. With parameters: $n = 20.000$, step size=60, $\beta = 4$, $k = 50$, $D_{min,robots} = 5$, $D_{min,O1-4} = 5$ and $D_{min,O5} = 10$.

We can observe that robot $R0$ starts from position (0.0, 0.0) in the workspace and travels to green area $l3$. Robot $R1$ starts from position (100.0, 0.0) in the workspace and travels to green area $l2$. Robot $R2$ starts from position (0.0, 100.0) in the workspace and travels to green area $l4$. And finally, robot $R3$ starts from position (100.0, 100.0) in the workspace and travels to green area $l5$. As mentioned in the caption, all of the starting positions are represented by zeros in the workspace.

Table 4.2: Performance of simulation in Figure 4.3b.

Total time	Path cost
1h44m17s71	550.25

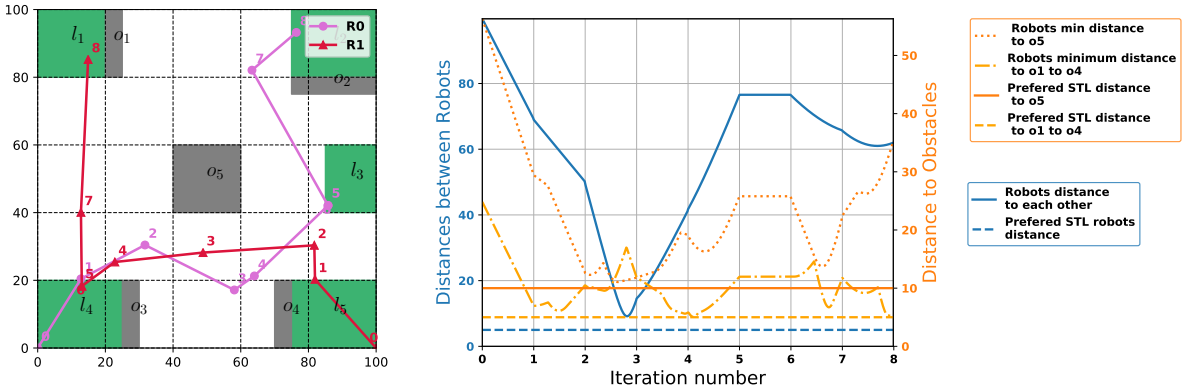
It is possible to observe that for four robots the developed algorithm takes more time to reach a good result. As mentioned above, the higher the number of robots, the harder it is to find synchronized trajectories between them. To generalize, when the number of robots increases, the computational time becomes considerably larger. This approach is not scalable. Centralized motion planning implies finding a simultaneous solution suiting all robots, which becomes more and more difficult with the increase in the number of robots. An alternative approach would be to use a decentralized solution, where the computational time is considerably reduced [36]. However, the trajectories tend to be more asymptotically optimal comparing to decentralized solutions since the robots' configurations are all chosen at the same time, therefore there are no dependencies between robots when selecting paths. This means that there are no prioritization between robots, the trajectories for all robots are defined simultaneously.

4.3.3 Two Robots - Two Goals Trajectory

The following case presents two robots' trajectories where robots need to reach two different green areas. The parameters used for this simulations are presented in the caption of Figure 4.4.

For this simulation, a minimum of 4.000 iterations was chosen in order to assure an asymptotically optimal result. 2.000 iterations are used for each goal per robot.

The resulting trajectories and environment distances of the simulation are presented below. We observer robot $R1$ starts at position (0,0, 0.0) and firstly passes by $l3$ while robot $R2$ starts at its initial position (100.0, 0.0) and passes by $l4$. After this, robot $R1$ goes to its final green area $l2$ while robot $R2$ goes to $l1$. Moreover, both STL formulas concerning the minimum preferred distances to obstacles and to robots are respected. The initial positions are marked with zeros and the different steps taken in the trajectory are enumerated.



(a) Two robots' simulated trajectories. **(b)** Robots' minimum distance to obstacles and robots.

Figure 4.4: Two robots' trajectories with two goals where the initial positions are marked with zeros and the order of states travelled is indicated. With parameters: $n = 4.000$, step-size=50, $\beta = 4$, $k = 50$, $D_{min,O1-4} = 5$ and $D_{min,O5} = 10$.

The resulting performances are presented in Table 4.3 below. The computational time is superior to 10 minutes, however the path cost is very little and the trajectory for both robots is asymptotically optimal.

Table 4.3: Performance of simulation in Figure 4.4.

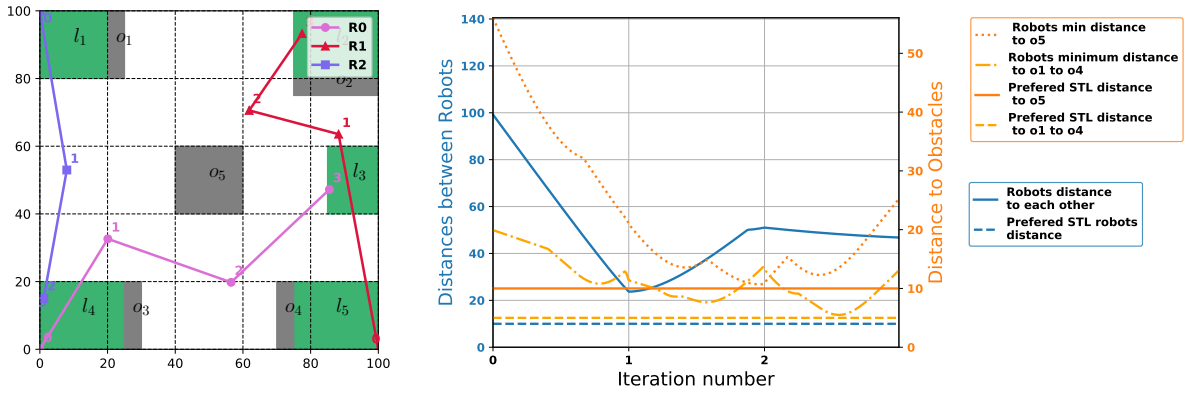
Total_time	Path cost
12m46s04ms	350.99

The computational time increases linearly with the number of goals to reach. Therefore, the solution is scalable when considering the number of goals instead of robots.

4.4 Optimization of the Algorithm

From the previous figures and simulations, we observe that occasionally, robots end up moving to unnecessary states during their trajectory. For example in Figure 4.2c, robot $R1$ moves from state 3 to 4 instead of state 3 to 5, or in Figure 4.3a, $R3$ moves from state 1 to 2 instead of 1 to 5. This is justified by the fact that all samples taken by the algorithm are collective and contain a position for each robot. Every sample added to the Tree is associated with a cost considering all of robots position in this sample. When path selection is done, it selects the path with the lesser cost. This path may contain states like state 4 in the simulation of Figure 4.2c, which might be

worst for robot $R1$'s trajectory but is good for $R0$ and $R2$' trajectories and does not disrespect STL specifications, having therefore a smaller cost.
However, after determining a preliminary trajectory and its respective cost, we can iterate the trajectories for each robot inside this global trajectory and determine if the cost can be optimised. We optimize it by seeing if deleting intermediate states in the robot's trajectory reduces the global cost. Furthermore, the robot will only move to the next states if it is not in risk of colliding with other robots. The result is presented in Figure 4.5.



(a) Three robots' simulated trajectories. **(b)** Robots' minimum distance to obstacles and robots.

Figure 4.5: Three robots' trajectories after optimization of trajectory in Figure 4.2c, where the initial positions are marked with zeros and the order of states travelled is indicated. With parameters: $n = 2.000$, step-size=60, $\beta = 4$, $k = 50$, $D_{min,O1-4} = 5$ and $D_{min,O5} = 10$.

Table 4.4: Performance of simulation in Figure 4.5.

Total_time	Path cost
5m11s54ms	322.65844837

From Figure 4.5a, we observe the number of states in the trajectories have been reduced. Indeed, $R1$ moves directly from state 0 to what used to be state 2 (and is now state 1) and move directly to what used to be state 5 (and is now state 2). The same happens for robots $R2$ and $R3$. We can observe from Figure 4.5b that the STL preferences are still being respected and that the number of states (iteration number) as been reduced from 6 to only 3 and the total cost has also been reduced. The rest of the trajectories shown previously in this section have also been optimised and their

simulation trajectories and costs are shown in Appendix section A.4. We observe that for all cases, the cost is reduced and the collision avoidance algorithm continues to function.

4.5 STL Impact on Motion Planning

The following simulations expose the STL impact on motion planning in multiple environments with the use of motion behavior specifications.

4.5.1 Testing STL Robustness Effect on Distance Between Robots - Minimum Distance

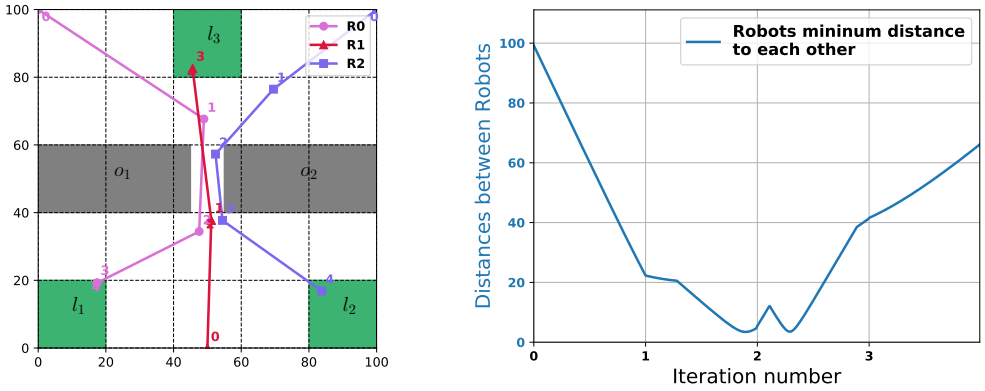
The following simulation includes three robots who need to pass by a single narrow area in order to reach their goal areas. Robot $R0$ starts from position (0.0, 100.0) in the workspace and travels to green area $l1$. Robot $R1$ starts from position (50.0, 0.0) in the workspace and travels to green area $l3$. And finally robot $R2$ starts from position (100.0, 100.0) in the workspace and travels to green area $l2$. All of the starting positions are represented by zeros in the trajectories. The parameters used for this simulation are presented in Table 4.5. The results on the performance are presented in Table 4.6.

Table 4.5: Parameters of simulations in Figures 4.6 and 4.7.

Iterations	k-nearest	Step size	$*\beta$	*STL distance to	
				o_1 & o_2	robots
8000	60	100	4	0	20

* *For the simulation with STL constraints.*

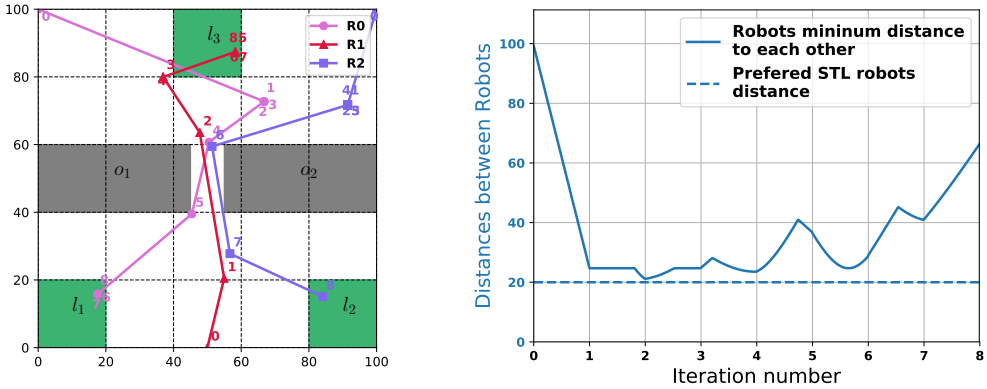
As seen in Figure 4.6, when STL preferences are not applied, the motion planning algorithm prevents them from colliding with each other. However, the distances between robots take very small values, being dangerous in real-case scenarios.



(a) Robots' trajectories without STL. **(b)** Minimum distances to robots.

Figure 4.6: Three robots' trajectories with no STL specifications where the initial positions are marked with zeros and the order of states travelled is indicated.

STL robustness fixes this by essentially "advising" robots to stay at a certain distance from each other. In the following simulation in Figure 4.7, robots are respecting the 20 meters minimum distance proposed by the STL formulas.



(a) Robots' trajectories with STL. **(b)** Minimum distances to robots.

Figure 4.7: Three robots' trajectories with STL specifications where the initial positions are marked with zeros and the order of states travelled is indicated.

Table 4.6: Performances of simulations in Figures 4.6 and 4.7.

STL	Total_time	Path cost
No	45m34s25	328.51
Yes	34m27s43	391.43

From the results obtained, we observe the trajectory lengths are bigger when STL constraints are used, and therefore the cost is higher. This is explained by the alternatives paths robots have to take to be separate from each other.

4.5.2 Testing STL Robustness Effect on Distance to Obstacles and Robots

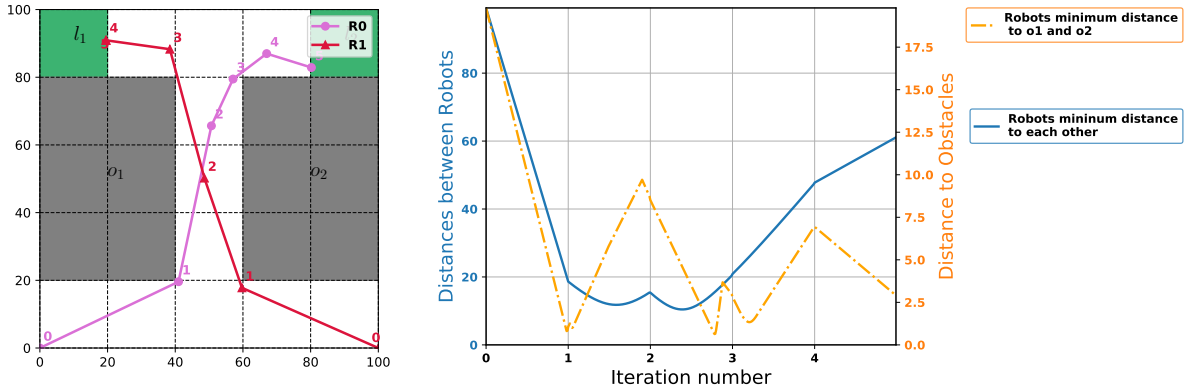
In this scenario, two robots need to travel within a thin corridor between two obstacles. What is aiming to be tested here is the capability of the STL robustness to influence the motion planning into selecting a path that respects a minimum of 5 meter distance to the obstacles, and also a minimum of 10 meters distance between robots. Robot $R0$ starts from position (0.0, 0.0) in the workspace and travels to green area $l2$. Robot $R1$ starts from position (100.0, 0.0) in the workspace and travels to green area $l1$. All of the starting positions are represented by zeros in the trajectories. The parameters used for this simulation are presented in Table 4.7. The results on the performance are presented in Table 4.8.

Table 4.7: Parameters of simulations in Figures 4.8 and 4.9.

Iterations	k-nearest	Step size	$*\beta$	*STL distance to	
				o_1 & o_2	robots
5000	50	60	4	5	10

* *For the simulation with STL constraints.*

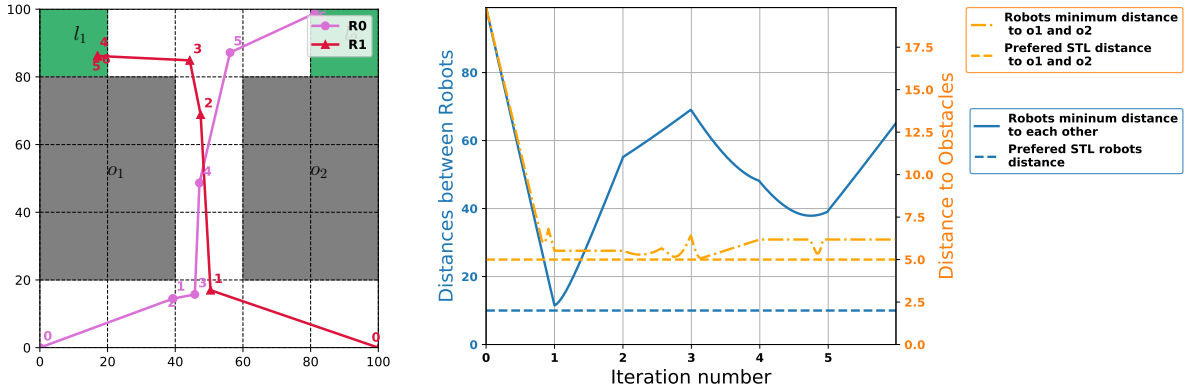
The simulation running without STL constraints is illustrated in Figure 4.8, where we observe robots get really close to the obstacles in order to have the lowest possible cost.



(a) Robots' trajectories without STL. **(b)** Minimum distances to obstacles and robots.

Figure 4.8: Simulation of two robots' trajectories without STL where the initial positions are marked with zeros and the order of states travelled is indicated.

In the simulation with STL constraints in Figure 4.9, two robots need to travel within a long narrow corridor between two obstacles while having to maintain a minimum distance from each other of 10 meters, and to each obstacle 5 meters.



(a) Robots' trajectories with STL. **(b)** Minimum distances to robots and obstacles.

Figure 4.9: Simulation of two robots' trajectories with STL constraints where the initial positions are marked with zeros and the order of states travelled is indicated.

Table 4.8: Performances of simulations in Figures 4.8 and 4.9.

STL	Total_time	Path cost
No	4m40s12	270.80
Yes	8m30s72	296.58

By comparing Figure 4.8 with Figure 4.9, it can be observed that when STL constraints are used, the trajectories are more distant to obstacles and to other robots. In fact, in Figure 4.9a the robots maintain a centered trajectory between obstacles to avoid breaking the user's preferences. In addition, R_0 takes more time to move further in its trajectory in order to give space for R_1 to advance. According to the results in Table 4.8, in exchange for safer trajectories, STL increases the computational time and cost performance.

4.5.3 Testing STL Robustness Effect on Distance to Robots - Maximum Distance

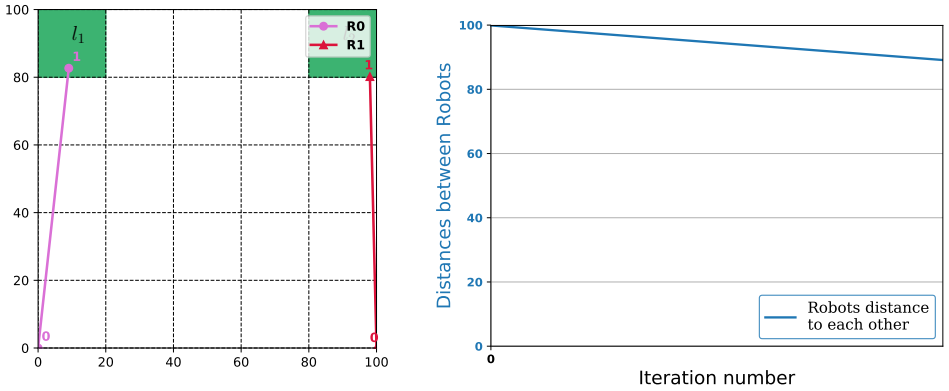
In this scenario, two robots need to travel with the minor cost to their respective goals while maintaining a maximum distance between each other. Robot $R0$ starts from position (0.0, 0.0) in the workspace and travels to green area $l1$. Robot $R1$ starts from position (100.0, 0.0) in the workspace and travels to green area $l2$. All of the starting positions are represented by zeros in the trajectories. The parameters used for this simulation are presented in Table 4.9. The results on the performance are presented in Table 4.10.

Table 4.9: Parameters of simulations in Figures 4.11 and 4.10.

Iterations	k-nearest	Step size	$*\beta$	*STL distance to robots
8000	30	70	4	10

* *For the simulation with STL constraints.*

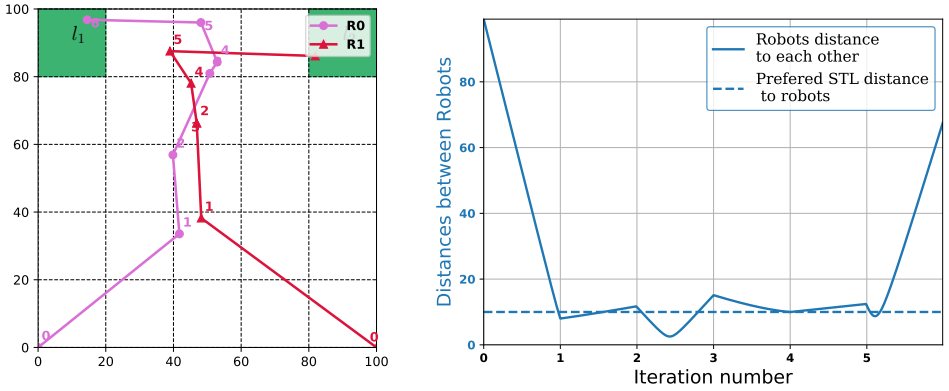
In Figures 4.10, the simulation without STL constraints shows two robots wanting to reach their goals with the minimum path cost. Therefore, they travel in straight lines to reach their final positions and have no influence on each other.



(a) Two robots' trajectories. **(b)** Robots' distances to each other.

Figure 4.10: Simulation of two robots' trajectories without STL constraints where the initial positions are marked with zeros and the order of states travelled is indicated.

In the simulation with STL constraints in Figures 4.11, two robots travel to their goal areas with the minimum necessary path length, while having to maintain a maximum distance of 10 meters from each other.



(a) Two robots' trajectories. **(b)** Robots' distances to each other.

Figure 4.11: Simulation of two robots' trajectories with STL constraints where the initial positions are marked with zeros and the order of states travelled is indicated.

Table 4.10: Performances of simulations in Figures 4.11 and 4.10.

STL	Total_time	Path cost
No	10m15s51	168.11
Yes	17m09s92	312.3

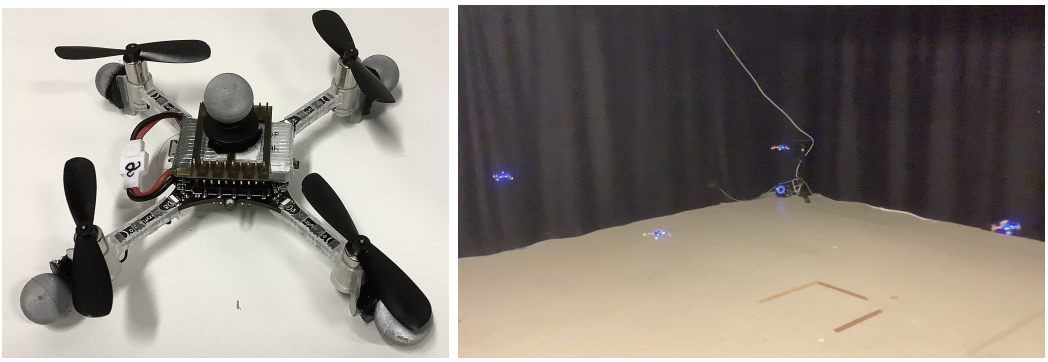
From Figure 4.11b, it is possible to observe the distance between robots tries to maintain a distance of exactly 10 meters since it also wants to maintain the cost as small as possible. We do not want robots to travel unnecessary distances to be closer to other robots than required since it means that they would have to travel longer trajectories than necessary. In Table 4.10, we can notice the cost for the selected path when using STL almost doubles due to initial and final parts of the trajectories where the robots are apart from each other until they are about 10 meters apart.

When comparing the developed solution with the one without STL restrictions, we conclude the use of STL restrictions prevents robots from being too close from each other or from obstacles in the surroundings, decreasing the chances of collision. It allows robots to travel within bigger distances from each other and from obstacles in the surroundings. Violations of the preferred distances only occur when necessary, i.e. when there is no better solution. The utility of STL is especially shown in particular cases such as in narrow corridors or when many robots have to pass in the same limited space to reach their goals. STL gives guidance to the robots' behaviors, making sure they respect its preferences to ensure their trajectory is safe. Using STL is also important when implementing the solution in the real world. Indeed, it allows margins of error in the robots' translations without risking collisions between robots as well as between robots and obstacles.

Chapter 5

Hardware Experiments and Discussion

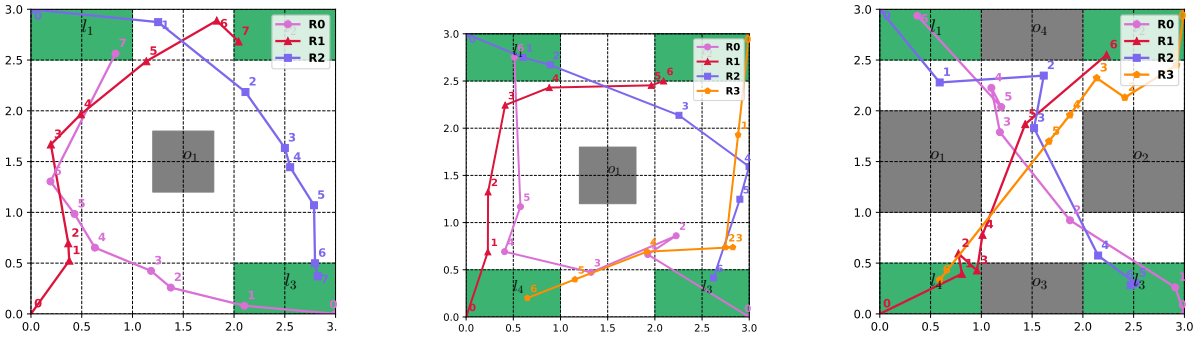
In this section, we will focus on implementing our developed solution in real-life through different ways. Since these experiments were conducted before the optimization process was developed, the simulations used for the real-life trajectories are not optimized. Crazyflies (CF) nano-quadrocopters with versions 2.0 and 2.1, as in Figure 5.1a, will be used to conduct these experiments. CFs are very convenient for these experiments. They have an available open-source software and they are not very expensive. In addition, their small size makes them less dangerous to work with and allows researchers to work with multiple CF at a time in small environments. For the experiments, we use the internal PID controller of the CF to which we send the set points of the positions we want to reach and at which velocity. Each CF has a unique pattern made with round markers (see Figure 5.1a) to be distinguished from other CFs. This way they are easily differentiated by the Mocap measurement system which is used to correct their position estimation. The Mocap system used is installed in the PMIL laboratory at KTH university and uses cameras and a Kalman Filter to correct the CFs positions. In Figure 5.1b, we can see a picture of an experiment of four robots running inside PMIL.



(a) **(b)**

Figure 5.1: Pictures of (a) a CF Nano-quadrocopter and (b) an experiment with four CF running in PMIL

There are two main complications in transforming the developed motion planning from simulation to real life. The first one consists in defining the velocities each robot needs to have when translating from one configuration(or state) to another. In fact, in the simulations the robots reach their respective configurations at the same time. This simplifies the collision avoidance algorithm, which compares the distance between robots while they travel between two consecutive configurations. Therefore, and to avoid collisions in real life, the experiments consider this as well. The second complication consists in smoothing the trajectories in order to avoid overshooting by the system, which is caused by drastic turns or high velocities. The trajectories used to test these experiments are trajectories with three or four robots: Figure 5.2a, Figure 5.2b. To test the collision avoidance algorithm in critical situations, we also test a simulation with four robots where each of their respective goals are positioned strategically to have multiple trajectories whose paths cross each other, Figure 5.2c. .

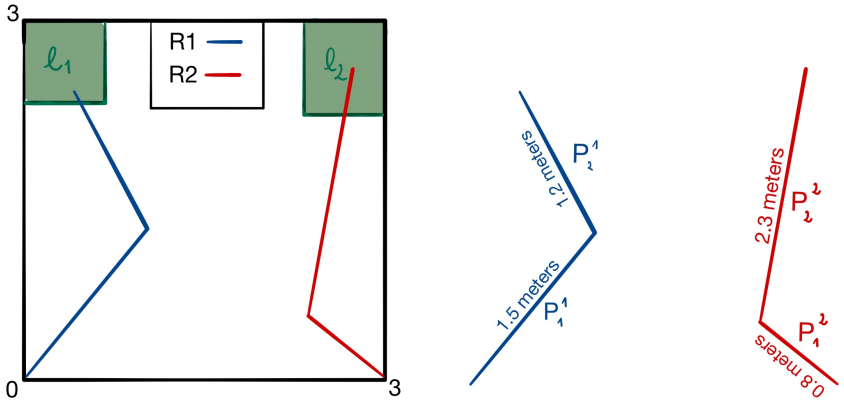


(a) **(b)** **(c)**

Figure 5.2: Simulations of trajectory with (a)three and (b)(c)four robots where the initial positions are marked with zeros and the order of states travelled is indicated.

We started by implementing a simpler version. The default velocity of the CFs is set to 1 m/s. When moving from state to state in the trajectory, we calculate the amount of time the biggest translation within all robots needs to be travelled. We conclude each robot needs to reach their next configuration within this interval of time and wait in the correspondent configuration until the end of this interval. After this interval is over, the next translations starts with the same process. This ensures all robots start each translation at the same time like in the simulations, minimising the risk of collision. The experiments using this version for controlling the position of three and four robots, as in simulation in Figure 5.2, are presented in Figures 5.4c,5.4d, in Figures 5.5c,5.5d and in Figures 5.6c, 5.6d.

The second version implemented is the *tracking version*. In this version we determine the longest path between all robots for each translation. We set a velocity of 1 m/s to the robot with the longest translation and determine the time it needs to traverse it. This time will be set as the ending time for all robots' translation between two configurations. Imagine the example below in Figure 5.3:



(a) **(b)**

Figure 5.3: Example of (a) a trajectory with two Robots R_1 and R_2 and (b) the information on their translations.

R_1 and R_2 have to travel across two translations, P_1 and P_2, before reaching their goal areas, l_1 and l_2. As observed, in the first translation we have $P_1^1 > P_1^2$ which means the robot with the biggest translation, R_1, will be travelling at a velocity of 1m/s. We estimate its arrival time to be at T = 1.5 seconds, as explained in equation (5.1).

$$T = \frac{\max(d_{P_1^1}, d_{P_1^2})}{V} = \frac{1.5\text{m}}{1\text{m/s}} = 1.5\text{s} \tag{5.1}$$

Where $d_{P_1^1}$ and $d_{P_1^2}$ correspond to the lengths of P_1^1 and P_1^2, respectively. Consequently, R_2 will also have to travel across its translation in 1.5 seconds. In order to do this, as soon as the CF is available for receiving messages, we send a new set of Set-Points corresponding to the position the robot should have at a specific moment. These Set-points are given by equation (5.2), where c_1 corresponds to the starting configuration of the translation and c_2 to the following configuration.

$$P(t) = \frac{c_2 - c_1}{T} \times t + c_1 \tag{5.2}$$

The results of the experiments conducted by running Figure 5.2's trajectories in the CFs with the *tracking version* are presented in Figures 5.4e,5.4f, in Figures 5.5f,5.5e and in Figures 5.6e, 5.6f.

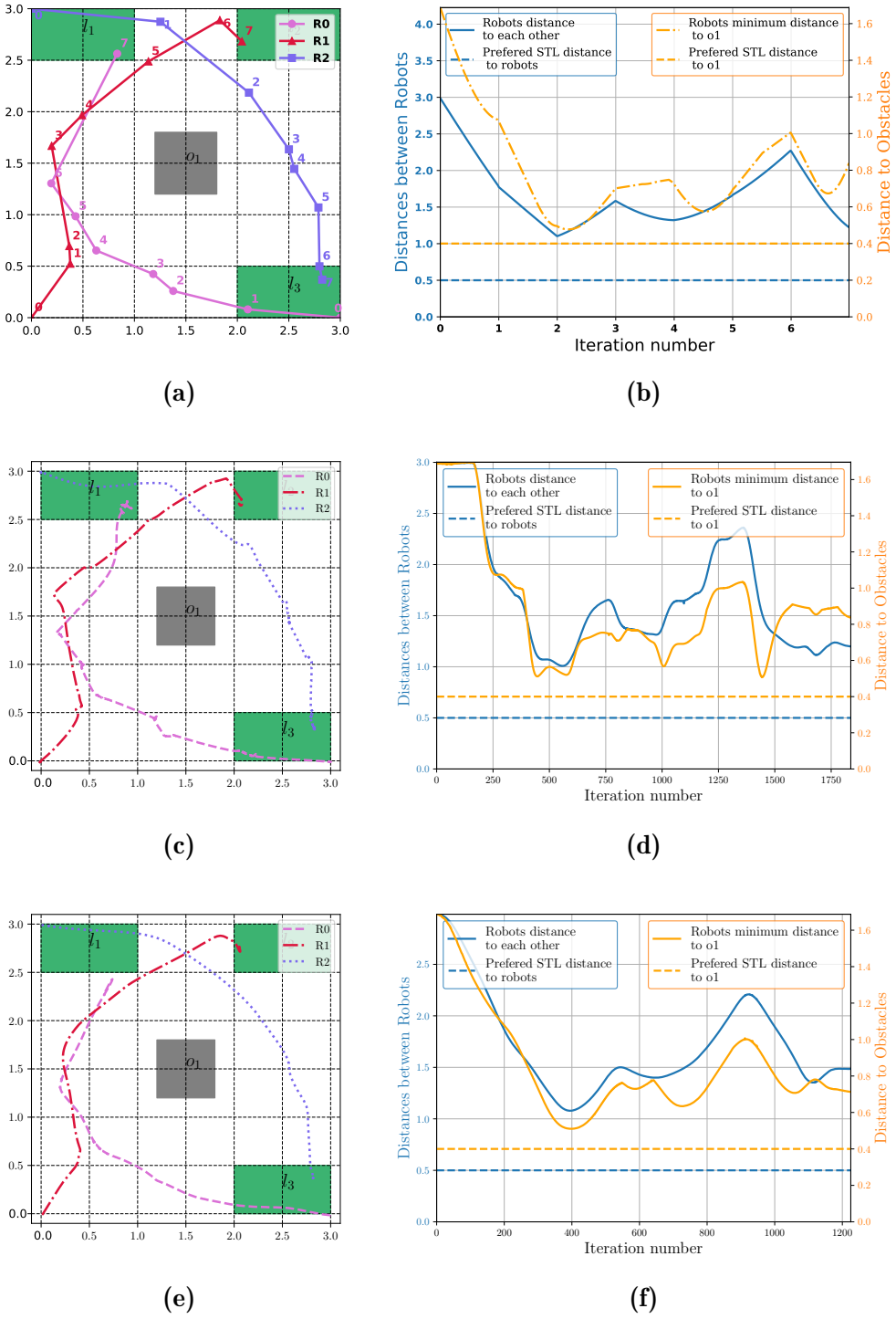


Figure 5.4: Trajectories with three robots in (a)(b)simulations, (c)(d)reality with simplest version and (e)(f)reality with tracking version. With $\beta = 4$, $n = 5000$, $k = 80$, $D_{min,robots} = 0.5$, and $D_{min,o1} = 0.4$.

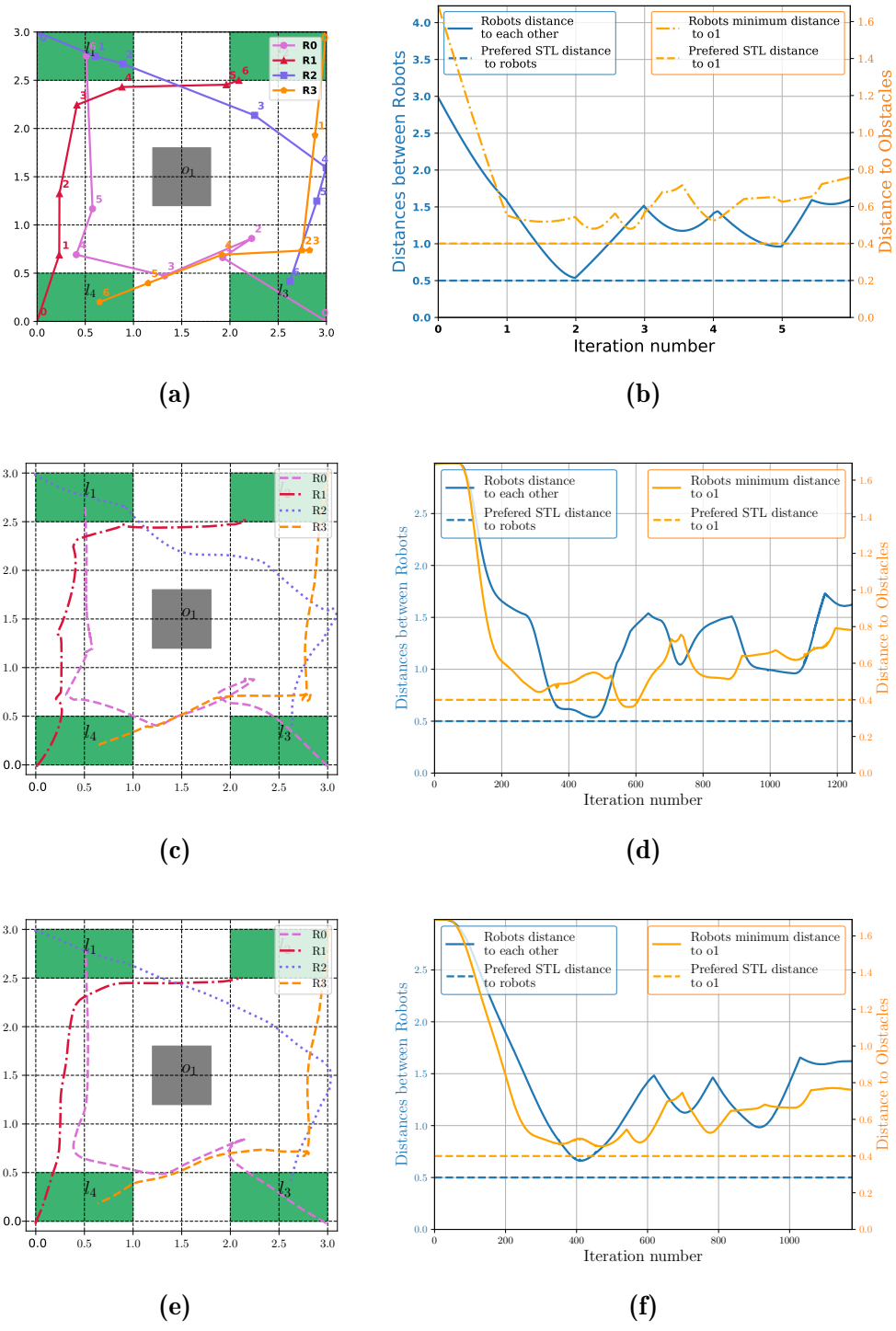


Figure 5.5: Trajectories with four robots in (a)(b)simulations, (c)(d)reality with simplest version and (e)(f)reality with tracking version. With $\beta = 4$, $n = 5000$, $k = 80$, $D_{min,robots} = 0.5$, and $D_{min,o1} = 0.4$.

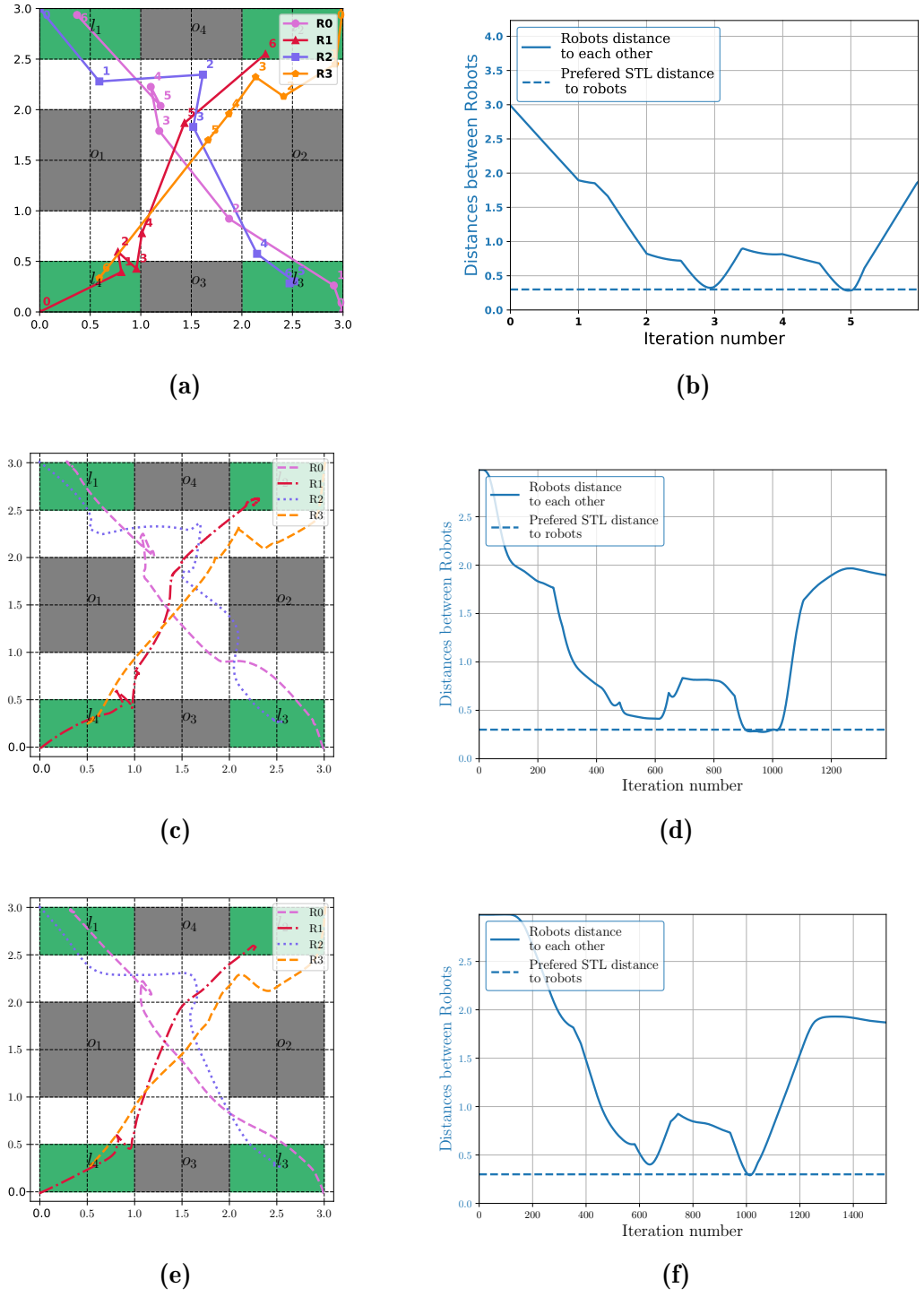


Figure 5.6: Trajectories with four robots testing the collision avoidance with $\beta = 4$, $n = 5000$ and $k = 50$, $D_{min,robots} = 0.4$. The initial positions are marked with zeros and the order of states travelled is indicated. In simulation (a)(b) and in reality with the (c)(d) simpler version and the (e)(f) tracking version.

By comparing the simplest version with the tracking version, we conclude the tracking version is more loyal to the simulations. The distances to obstacles are more similar to theory. Meanwhile in the simpler version, it is more likely that the trajectories breach the obstacles' spaces and endanger the robots.

To conclude, using the tracking version for the real world implementation reflects accurately the same behaviour as demonstrated in the simulation. Additionally, the distances between obstacles and robots were also proven to be practically identical to theory. Finally, the small delay when sending positions to the CF controllers allowed a better transition from configurations to configuration, resulting in smoother trajectories than with the simpler version.

Chapter 6

Sustainability and Ethics

Drone technology has been a subject of rapid development in the past decades. Today drone applications can be seen in many different sectors. UAVs are becoming omnipresent in industries like the military, the post, agriculture, leisure and many more. One key characteristic of drones is that they are able to travel fast and to reach areas that are uneasily accessed by humans. A particular use of drones is the surveillance and security control application. For instance, drones are able to survey big areas such as mountains or post-disaster zones with a goal of searching for people in danger. In fact, drones are directly allied with sustainability issues which can include enhancing environmental monitoring with fire control, to managing agricultural resources like water for example. However, there is a thin line between surveillance control and security and/or privacy breach. Taking into account that nowadays drones are accessible for all, many rules need to be established in order to assure the safety of others. In fact, drones can be used to monitor security in public places, road traffic, etc. Gathering data about individuals without their consent can already be considered as an issue of privacy violation. The same issue applies when doing something as simple as taking pictures in a public area. If these pictures feature people present in the environment, publishing them may be considered illegal. In this work, by combining multi-agent motion planning with STL constraints, we are allowing the drones to follow a more customized trajectory. We can reinforce collision avoidance by making robots stay further away from obstacles and other robots in the environment. Finally, by constructing asymptotically optimal solutions, we limit the battery cost by avoiding unnecessarily long trajectories.

Chapter 7

Conclusions

In this thesis, a multi-agent motion planning approach was developed using RRT* trajectory planning and with STL to constrain the robots' motion behaviours. A user can personalize the motion planning by using STL formulas to establish preferences in the robots' trajectory. Users can decide how important it is for the robots to respect their preferences by tuning the STL β parameter. The motion planning takes into account the preferences and is influenced to select trajectories which respect them. From this work, we conclude that using STL constraints and STL robustness in the motion planning, changes the robots' motion behaviors into respecting the user's specifications. We have seen that robots usually opt for longer paths in order to avoid travelling too close to obstacles and to other robots. This is what is seen in Figure 4.9, where multiple robots circulating in an environment need to travel within a narrow corridor. To respect STL constraints, they travel one after the other to avoid collision. In other cases, as in Figure 4.11, robots travel at a maximum and minimum distance from each other instead of opting for a direct quicker path.

Moreover, STL constraints are only violated when the algorithm can not identify a solution, or in this case a trajectory, which respects them. This is the case of the simulation represented in Figure 5.6, where four robots have to cross each other's paths in order to reach their goals. It can be seen that small violations of user's preferences occur since it is not attainable to find solutions which compensate the current cost and would respect entirely the STL constraints.

Finally, the addition of STL constraints is followed by an increment of the computational time. This is due to the extra calculations of STL violations and robustness values which are used to influence the cost of each path. Robustness calculation includes distance comparisons between robots and obstacles as well as between robots. As a result, an increase in the algorithm's processing time is observed, which is compensated with the preservation of the user's preferences and with asymptotically optimal

trajectories.

Chapter 8

Future Work

As mentioned before, one of the main problems with this approach is the non-scalability. Although it is partially corrected with the optimization in section 4.4, the solution remains mostly centralized and non-scalable for a higher number of robots. One possible solution would be to implement a completely decentralized approach, which is the case in [36]. The robots have information on their position and other robots' dynamics and the algorithm focuses in two steps. In the *individual component*, each robot determines a finite or non-finite trajectory, all robots' trajectories are compared and the one with the lesser cost keeps it while others adapt accordingly, in the *interaction component*. In this second step, robots exchange messages with the *winning* trajectory and reconstruct their trajectory accordingly. The process is then repeated.
The lack of scalability can also be improved by varying the *stepsize* according to the evolution of the RRT*'size during motion planning. When the Tree still displays a small dimension, finding a set of configurations that can reach the tree in less than a certain distance can be challenging. This is why, in the beginning of the motion planning, the *stepsize* should be bigger to allow the configurations to find nearby nodes and to be added to the tree. As the nodes in the Tree increase, the Tree starts filling the workspace, which means it is easier for new configurations to find nearby nodes. Therefore the *stepsize* can be reduced to avoid collecting an unnecessarily big number of nearby nodes which analysis increases the computational time.
This thesis project is an off-line approach for supervision and monitoring of defined environments where all obstacles and limits are known beforehand. For dynamic or unknown environments, it is interesting to change it to an on-line approach. This way, robots are able to make changes in their trajectory if dynamic objects appear in their way. It can also be interesting for exploration approaches like in [5] where the position of the robot is defined according the features of the environment.
In this thesis work, STL robustness is calculated with the value of the most violating

distance between robots and robots, and robots and obstacles. It does not take into account if a configuration is breaking more than one preference at a time. This means that if one robot is too close to an object but another is even closer to an object, it will only consider the most violating distance between both cases. This may seem unfair if in another case, a robots is at the same most violating distance from an obstacle but all other robots are respecting the STL preferences. The STL robustness will be the same as in the previous case, not differentiating one case is better than the other. By using the average value of violating and non-violating distances between robots and between robots and obstacles, we take into account if more than one robots is not respecting the STL formulas.

Bibliography

[1] Minguez J., Lamiraux F., and Laumond JP. "Motion Planning Concepts". In: *Springer Handbook of Robotics*. Ed. by Heidelberg Springer Berlin. 2008, pp. 109–129.

[2] Elbanhawi M. and Simic M. "Sampling-Based Robot Motion Planning: A Review". In: ed. by IEEE. 2014.

[3] Minguez J., Lamiraux F., and Laumond JP. "Motion Planning and Obstacle Avoidance". In: *Springer Handbook of Robotics*. Ed. by Heidelberg Springer Berlin. 2008, pp. 827–852.

[4] Gasparetto A. et al. "Path Planning and Trajectory Planning Algorithms: A General Overview". In: *Mechanisms and Machine Science* (2015).

[5] S. Barbosa F., Jensfelt P. Duberg D., and Tumova J. "Guiding Autonomous Exploration With Signal Temporal Logic". In: *IEEE Robotics and Automation Letters* (2019).

[6] Honig W. et al. "Trajectory Planning for Quadrotor Swarms". In: *IEEE Transaction on Robotics* (2018).

[7] Bogg P., Beydoun G., and Low G. "When to Use a Multi-Agent System?" In: *Lecture Notes in Computer Science, vol 5357* (2008).

[8] Guo M., H. Johansson K., and V. Dimarogonas D. "Revising motion planning under linear temporal logic specifications in partially known workspaces". In: *IEEE International Conference on Robotics and Automation (ICRA)* (2013), pp. 5025–5032.

[9] Zvi Shiller. "Off-Line and On-Line Trajectory Planning". In: *Motion and Operation Planning of Robotic Systems*. Ed. by Springer International Publishing. 2015, pp. 29–62.

[10] Khaksar W. et al. "Application of Sampling-Based Motion Planning Algorithms in Autonomous Vehicle Navigation". In: *Autonomous Vehicle*. Ed. by Intech Open. 2016, pp. 21–35.

[11] Karaman S. and Frazzoli E. “Sampling-based algorithms for optimal motion planning”. In: *The International Journal of Robotics Research, vol. 30.* 2011, pp. 846–894.

[12] Martínez Alandes C. “A comparison among different sampling-based planning techniques”. In: 2014.

[13] Kala R. and Warwick k. “Planning of Multiple Autonomous Vehicles using RRT”. In: ed. by IEEE International Conference on Cybernetic Intelligent. 2011, pp. 20–25.

[14] D. Gammell J., S. Srinivasa S., and D. Barfoot T. “Informed RRT*: Optimal Sampling-based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic”. In: ed. by IEEE/RSJ International Conference on Intelligent Robots and Systems. 2014, pp. 2997–3004.

[15] Cap M. et al. “Multi-agent RRT*: Sampling-based Cooperative Pathfinding”. In: ed. by Autonomous agents and multi-agent systems. 2013, pp. 1263–1264.

[16] Hussain A. and Ayaz Q. “Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments”. In: *Robotics and Autonomous Systems* (2015).

[17] Chin T. “Robotic Path Planning: RRT and RRT*”. In: *A Medium Corporation* (2019).

[18] Noreen I., Khan A., and Habib Z. “A Comparison of RRT, RRT* and RRT*-Smart Path Planning Algorithms”. In: *IJCSNS International Journal of Computer Science and Network Security* (2016).

[19] Jaillet L., Cortes J., and Simeon T. “Transition-based RRT for Path Planning in Continuous Cost Spaces”. In: *Association for the Advancement of Artificial Intelligence* (2008).

[20] Bhattacharya S., Likhachev M., and Kumar V. “Multi-agent Path Planning with Multiple Tasks and Distance Constraints”. In: ed. by IEEE International Conference on Robotics and Automation. 2010.

[21] Eduardo Alonso. “From Artificial Intelligence to Multi-Agent Systems: Some Historical and Computational Remarks”. In: (2001).

[22] Vardhan Pant Y. et al. “Fly-by-Logic: Control of Multi-Drone Fleets with Temporal Logic Objectives”. In: *Theoretical Computer Science* (2009).

[23] J. Gainer Jr. J. et al. “Persistent Multi-Agent Search and Tracking with Flight Endurance Constraints”. In: *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference* (2018).

[24] Kivelevitch E. and Cohen K. “Multi-Agent Maze Exploration”. In: ed. by Journal of Aerospace Computing Information and. 2010.

[25] Kantaros Y. and M. Zavlanos M. “Temporal Logic Optimal Control for Large-Scale Multi-Robot Systems: 10400 States and Beyond”. In: *IEEE Conference on Decision and Control* (2018).

[26] C. Madsen et al. “Metrics for Signal Temporal Logic Formulae”. In: *IEEE Conference on Decision and Control (CDC)* (2018).

[27] O. Maler and D. Nickovic. “Monitoring temporal properties of continuous signals”. In: *In Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems* (2004). Ed. by Springer, pp. 152–166.

[28] Karen Leung K., Arechiga N., and Pavone M. “Back-propagation through STL Specifications: Infusing Logical Structure into Gradient-Based Methods”. In: *(submitted)* (2020).

[29] A. Donzé and O. Maler. “Robust satisfaction of temporal logic over realvalued signals”. In: *8th Int. Conf. Formal Model. Anal. Timed Syst.* (2010), pp. 92–106.

[30] Vasile C., Raman V., and Karaman S. “Sampling-based synthesis of maximally-satisfying controllers for temporal logic specifications”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems* (2017), pp. 3840–3847.

[31] Vasile C.-I. et al. “Minimum-violation scLTL motion planning for mobility-on-demand”. In: *IEEE International Conference on Robotics and Automation (ICRA)* (2017).

[32] Liu Z.and Wu B., Dai J., and Lin H. “Communication-aware Motion Planning for Multi-agent Systems from Signal Temporal Logic Specifications”. In: *American Control Conference (ACC)* (2017).

[33] Kantaros Y. and M. Zavlanos M. “Sampling-Based Optimal Control Synthesis for Multi-Robot Systems under Global Temporal Tasks”. In: *IEEE Transactions on Automatic Control* (2017).

[34] A. lusoy et al. “Optimality and Robustness in Multi-Robot Path Planning with Temporal Logic Constraints”. In: ed. by The International Journal of Robotics Research. 2013, pp. 889–911.

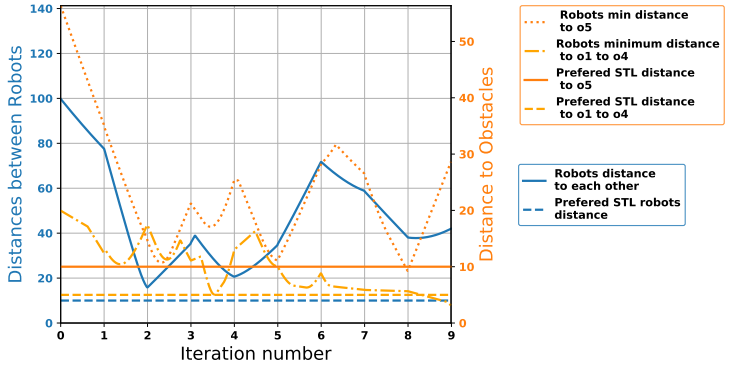
[35] Tumova J.and V. Dimarogonas D. “Multi-Agent Planning under Local LTL Specifications and Event-Based Synchronization”. In: *Automatica* (2016).

[36] R. Desaraju V. and P. How J. “Decentralized Path Planning for Multi-Agent Teams in Complex Environments using Rapidly-exploring Random Trees”. In: ed. by IEEE International Conference on Robotics and Automation. 2011, pp. 4956–4961.

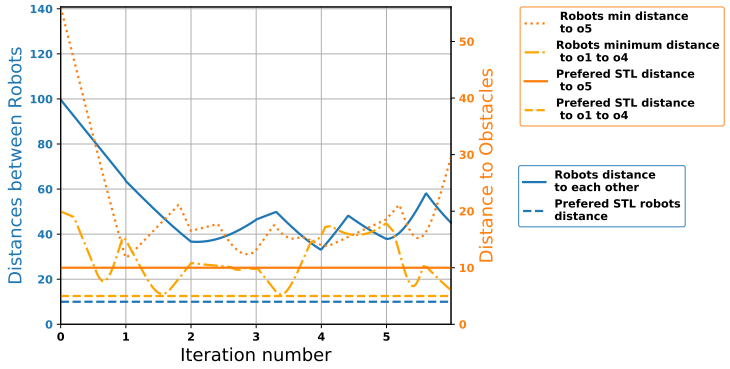
Appendix A

Appendix A contains more details about the some of the simulations in Section 4 and of the simulations done to evaluate the different values of parameters for the motion planning algorithm.

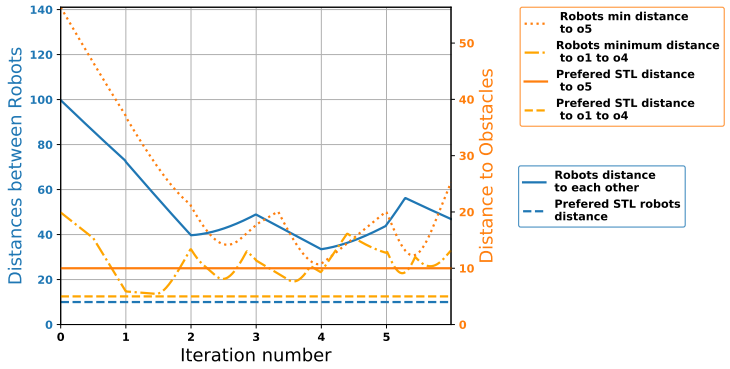
A.1 Distances to Robots and obstacles with different versions of the motion planning algorithm RRT*



(a) Robots' distances to each other and to obstacles w/ MA-RRT* simple version.



(b) Robots' distances to each other and to obstacles w/ MA-RRT* w/ Biased Sampling(BS).



(c) Robots' distances to each other and to obstacles w/ MA-RRT* w/ BS and k-nearest neighbors.

Figure A.1: Distances to obstacles and robots with different versions of the motion planning algorithm.

A.2 β Testing simulations with the developed MA-RRT*

Study of tuning parameter β

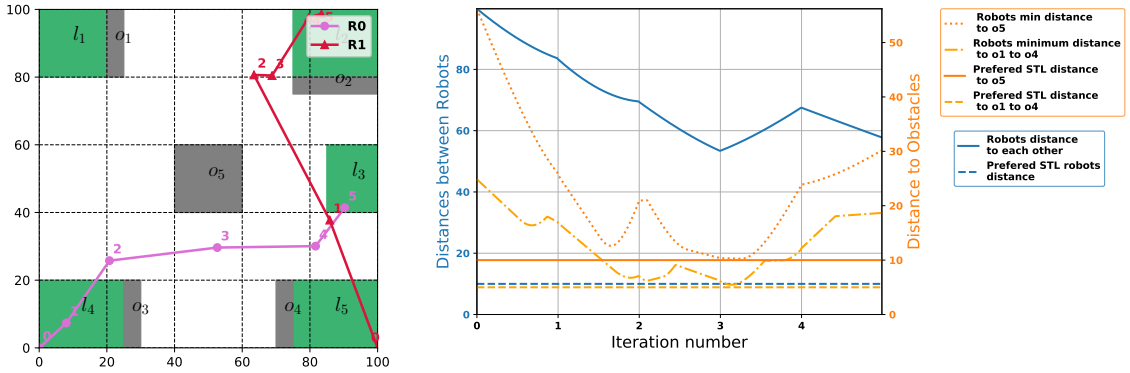
To test the β value, two robots will be used again to run the simulation. Robot R0 will start from position (0.0, 0.0) in the workspace and will travel to goal area $l3$. Robot R1 will start from position (100.0, 0.0) in the workspace and will travel to goal area $l2$. All of the starting positions are represented by black stars in the workspace. The parameters used for this simulation are present in Table A.1. Finally, the results on the performance are presented in the Table A.2.

Table A.1: Parameters for simulation in Figure A.2, A.3 and A.4.

Iterations	Step size	K-near	β	STL distance to		
				o_1 to o_4	o_5	robot
1000	60	100	β	5	10	10

In the following simulations, the β values which resulting performances are to be compared, are 0.4, 4, and 8.

A.2.1 Simulation with $\beta = 0.4$

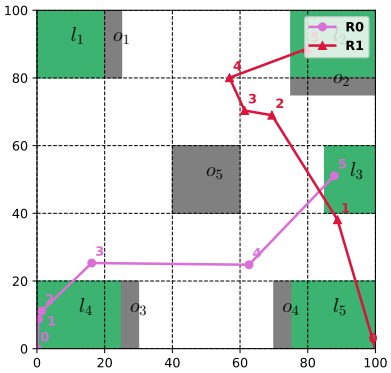


(a) Trajectories for two robots.

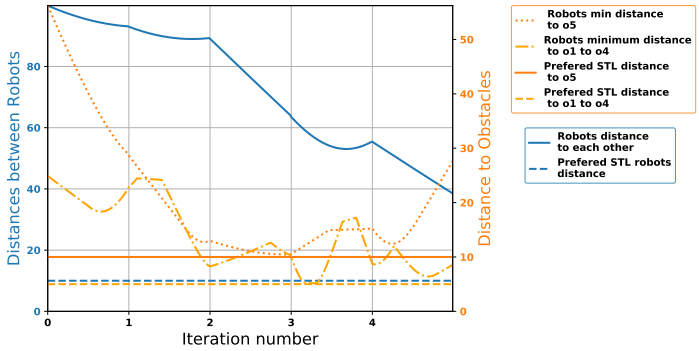
(b) Distance between Robots: $D_{min,robots}$=10, and to obstacles: $D_{min,o1-4}$=5, $D_{min,o5}$=10.

Figure A.2: Two robots' trajectories with $\beta = 0.4$, $n = 1000$, step size=60, k=100. The initial positions are marked with zeros and the order of states traveled is indicated.

A.2.2 Simulation with $\beta = 4$



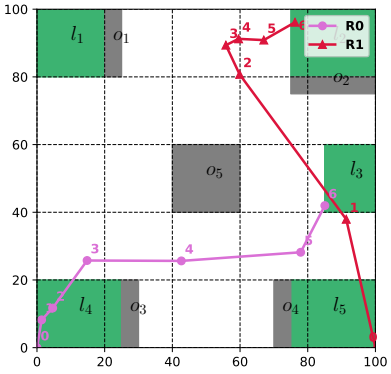
(a) Trajectories for two robots.



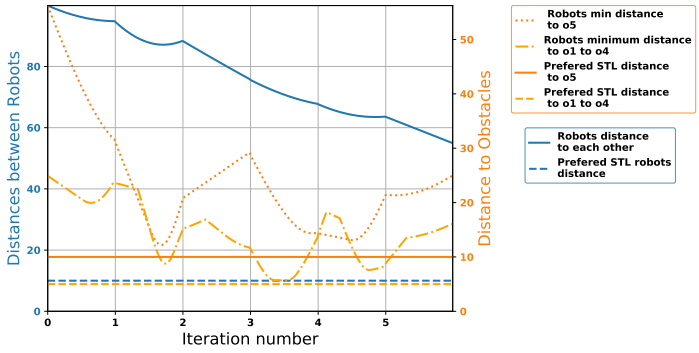
(b) Distance between Robots: $D_{min,robots}$=10, and to obstacles: $D_{min,o1-4}$=5, $D_{min,o5}$=10.

Figure A.3: Two robots' trajectories with $\beta = 4$, $n = 1000$, step size=60, k=100. The initial positions are marked with zeros and the order of states traveled is indicated.

A.2.3 Simulation with $\beta = 8$



(a) Trajectories for two robots.



(b) Distance between Robots: $D_{min,robots}$=10, and to obstacles: $D_{min,o1-4}$=5, $D_{min,o5}$=10.

Figure A.4: Two robots' trajectories with $\beta = 8$, $n = 1000$, step size=60, k=100. The initial positions are marked with zeros and the order of states traveled is indicated.

The respective performances of these simulations are presented in Table A.2.

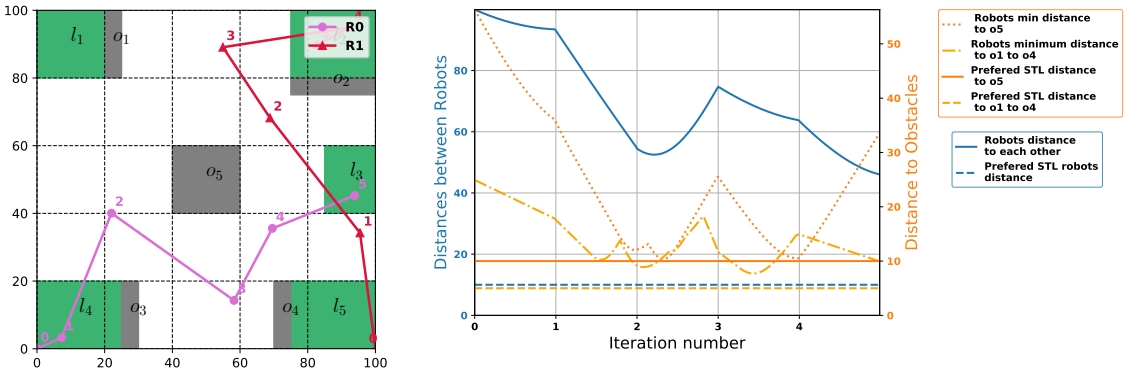
Table A.2: Performances of simulations for the different numbers of β.

β	Path cost
0.4	226.91
4	234.27
8	233.03

The time performance is not affected by the values of β, therefore it will not be taken into account for this study. For all of the β values studied, the STL preference are respected. When the beta value is considerably smaller, the effect of the STL robustness is less important. In other words, if the robustness value is not high enough to make a significant difference in the trajectory cost, no obvious distinction is made between paths respecting and violating STL formulas. It can be used to determine whether sometimes it is better to opt for a shorter path instead of a longer path which respect the STL preferences. If we choose to be more strict with the STL preferences (like in this thesis work), a higher value for beta should be chosen: $\beta = 4$. By examining Figure A.3, $\beta = 4$ shows to perfectly influence the trajectory planning into respecting the STL preferences. For $\beta > 4$ the trajectory does not show major improvements.

A.3 Iteration number n Testing simulations with the developed MA-RRT*

A.3.1 Simulation with $n = 100$

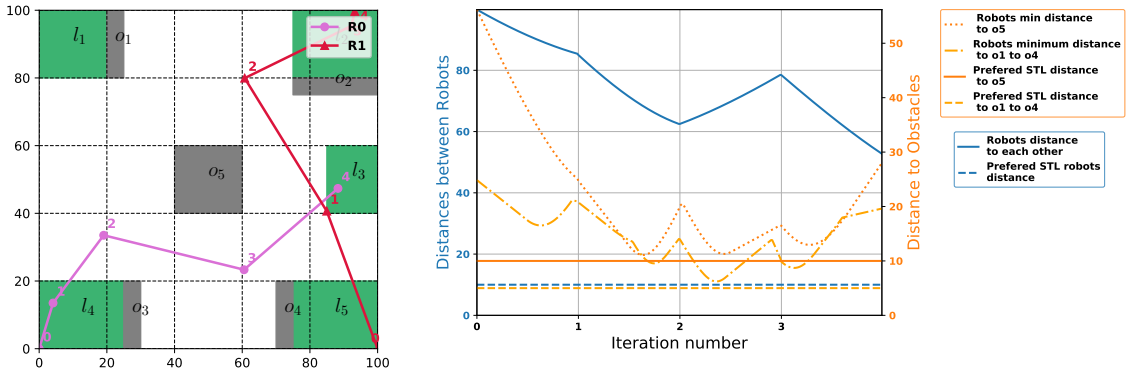


(a) Trajectories for two robots.

(b) Distance between Robots: $D_{min,robots}$=10, and to obstacles: $D_{min,o1-4}$=5, $D_{min,o5}$=10.

Figure A.5: Two robots' trajectories with $n = 100$, step size=60, $\beta = 4$, k=100. The initial positions are marked with zeros and the order of states traveled is indicated.

A.3.2 Simulation with $n = 500$

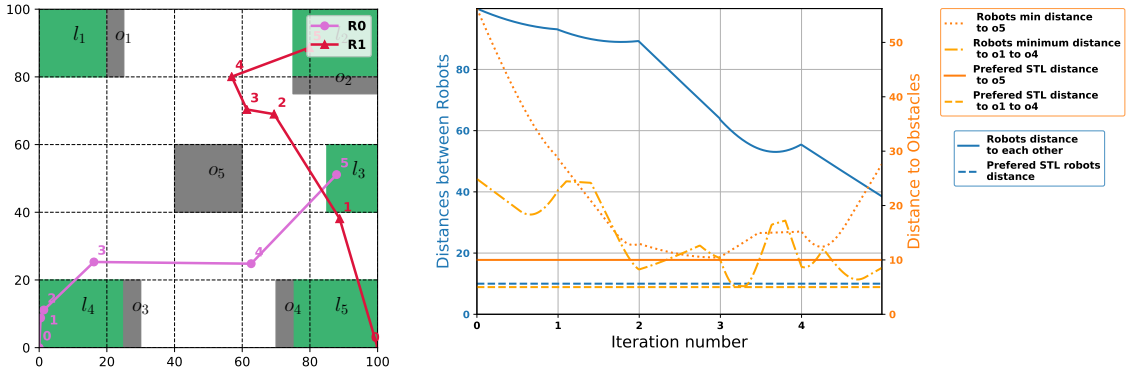


(a) Trajectories for two robots.

(b) Distance between Robots: $D_{min,robots}$=10, and to obstacles: $D_{min,o1-4}$=5, $D_{min,o5}$=10.

Figure A.6: Two robots' trajectories with $n = 500$, step size=60, $\beta = 4$, k=100. The initial positions are marked with zeros and the order of states traveled is indicated.

A.3.3 Simulation with $n = 1000$

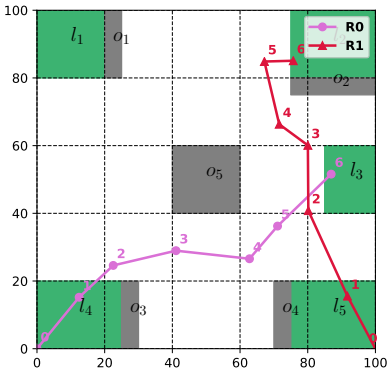


(a) Trajectories for two robots.

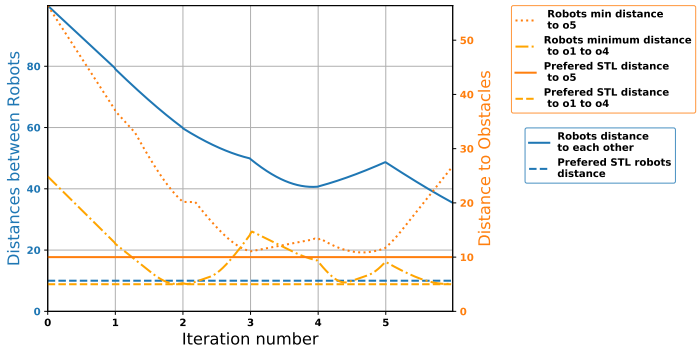
(b) Distance between Robots: $D_{min,robots}$=10, and to obstacles: $D_{min,o1-4}$=5, $D_{min,o5}$=10.

Figure A.7: Two robots' trajectories with $n = 1000$, step size=60, $\beta = 4$, k=100. The initial positions are marked with zeros and the order of states traveled is indicated.

A.3.4 Simulation with $n = 3000$



(a) Trajectories for two robots.



(b) Distance between Robots: $D_{min,robots}$=10, and to obstacles: $D_{min,o1-4}$=5, $D_{min,o5}$=10.

Figure A.8: Two robots' trajectories with $n = 3000$, step size=60, $\beta = 4$, k=100. The initial positions are marked with zeros and the order of states traveled is indicated.

The performances for the experiments above are presented in Table A.3.

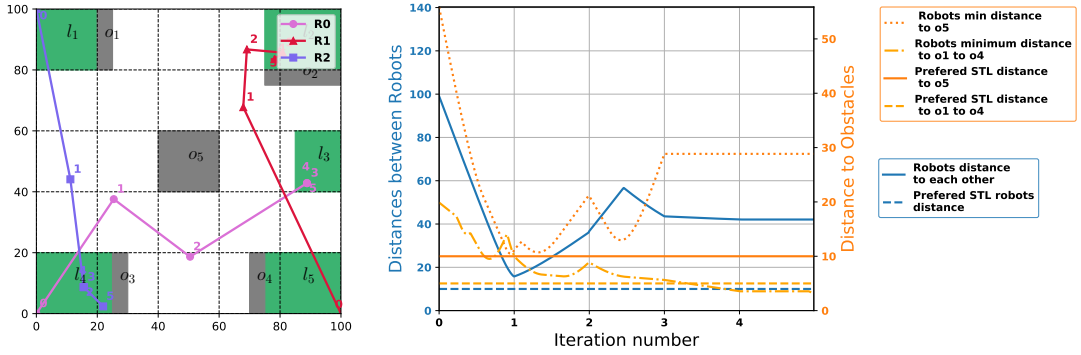
Table A.3: Performances of simulations for the different numbers of minimum iteration n before implementing k - nearest neighbours.

n	Total time	Path cost
100	00m04s33	294.21
500	00m42s49	247.53
1000	02m16s52	234.27
3000	08m40s10	211.85

It is important to mention that the algorithm takes more time to find trajectories if the workspace is more complex and has more obstacles, the computational time also increases with the number of robots implied. The cost decreases with the number of iterations. The more iterations we run on the algorithm, the better cost the trajectories found have. Adding the k - nearest nodes methods is essential to decrease the algorithm computational time.

A.4 Optimized trajectories of simulations presented in section chapter 4

A.4.1 Optimized trajectory of simulation in Figure 4.2a



(a) Trajectories for three robots. **(b)** Minimum distance to robots and obstacles.

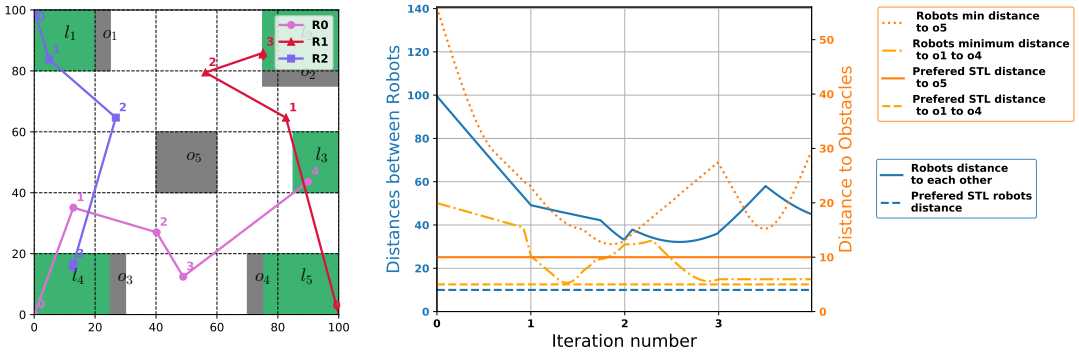
Figure A.9: Three robots' trajectories with simple MA-RRT* version with optimization. The initial positions are marked with zeros and the order of states traveled is indicated. With parameters: $n = 2.000$, step size=60, $\beta = 4$, $D_{min,robots} = 10$, $D_{min,O1-4} = 5$ and $D_{min,O5} = 10$.

The performances for the experiments above are presented in Table A.4.

Table A.4: Performances of simulations before, Figure 4.2a and after optimization, Figure A.9

Path Cost	
Before	After
75489.10	21252.44

A.4.2 Optimized trajectory of simulation in Figure 4.2b



(a) Trajectories for three robots. **(b)** Minimum distance to robots and obstacles.

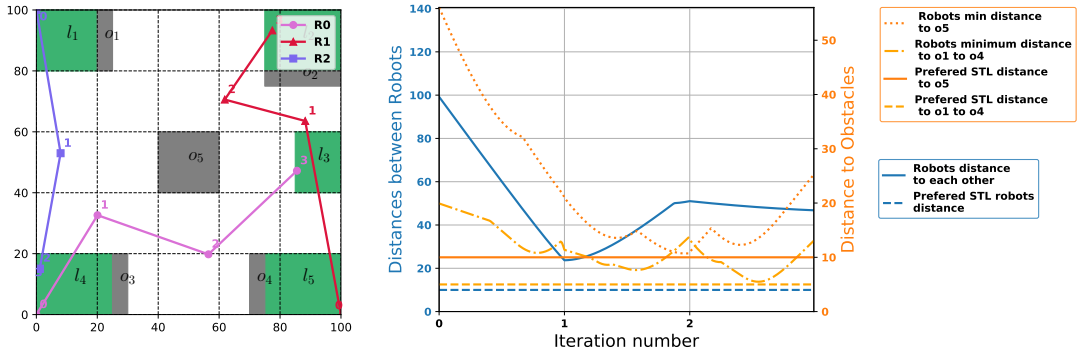
Figure A.10: Three robots' trajectories with simple ma-rrt* version with biased sampling with optimization. The initial positions are marked with zeros and the order of states traveled is indicated. With parameters: $n = 2.000$, step size=60, $\beta = 4$, $D_{min,robots} = 10$, $D_{min,O1-4} = 5$ and $D_{min,O5} = 10$.

The performances for the experiments above are presented in Table A.5.

Table A.5: Performances of simulations before, Figure 4.2b and after optimization, Figure A.10

Path Cost	
Before	After
380.72	340.54

A.4.3 Optimized trajectory of simulation in Figure 4.2c



(a) Trajectories for three robots. **(b)** Minimum distance to robots and obstacles.

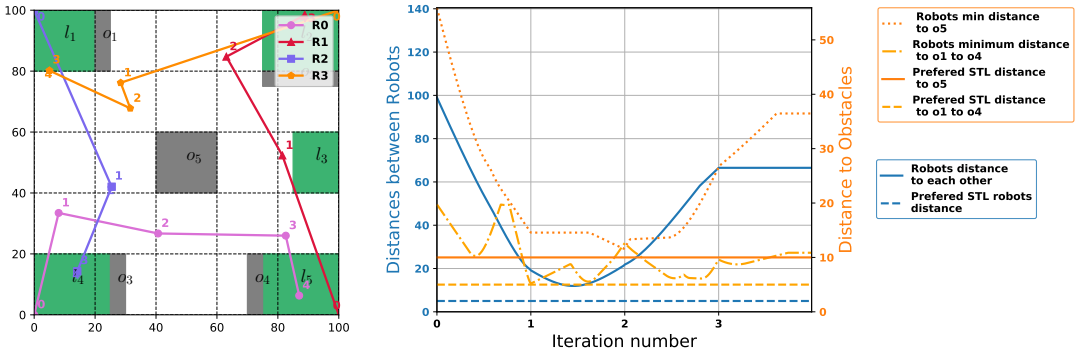
Figure A.11: Three robots' trajectories with simple ma-rrt* version with biased sampling and the K-nearest nodes method, with optimization. The initial positions are marked with zeros and the order of states traveled is indicated. With parameters: $n = 2.000$, step size=60, $\beta = 4$, $D_{min,robots} = 10$, $D_{min,O1-4} = 5$ and $D_{min,O5} = 10$.

The performances for the experiments above are presented in Table A.6.

Table A.6: Performances of simulations before, Figure 4.2c and after optimization, Figure A.11

Path Cost	
Before	After
365.8	322.66

A.4.4 Optimized trajectory of simulation with four robots in Figure 4.3



(a) Trajectories for four robots. **(b)** Minimum distance to robots and obstacles.

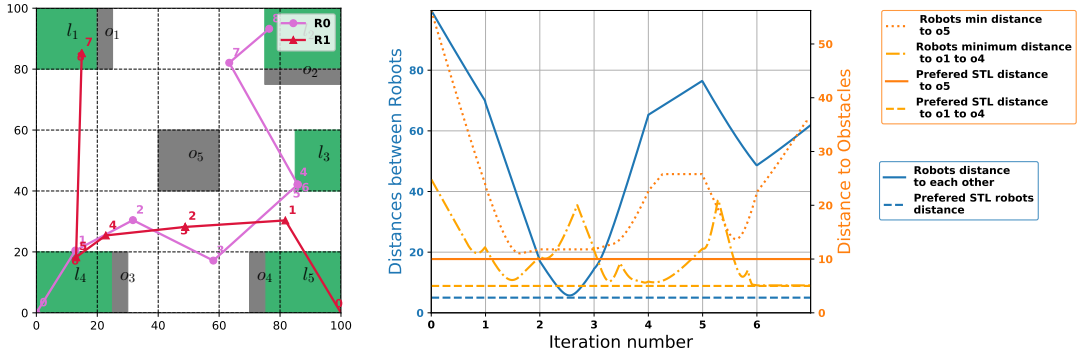
Figure A.12: Four robots' trajectories after optimization where the initial positions are marked with zeros and the order of states traveled is indicated. With parameters: $n = 20.000$, step size=60, $\beta = 4$, $k = 50$, $D_{min,robots} = 5$, $D_{min,O1-4} = 5$ and $D_{min,O5} = 10$.

The performances for the experiments above are presented in Table A.7.

Table A.7: Performances of simulations before, Figure 4.3 and after optimization, Figure A.12

Path Cost	
Before	After
550.25	458.65

A.4.5 Optimized trajectory of simulation with two robots in Figure 4.4



(a) Trajectories for two robots. **(b)** Minimum distance to robots and obstacles.

Figure A.13: Two robots' trajectories after optimization with two goals where the initial positions are marked with zeros and the order of states traveled is indicated. With parameters: $n = 4.000$, step-size=50, $\beta = 4$, $k = 50$, $D_{min,O1-4} = 5$ and $D_{min,O5} = 10$.

The performances for the experiments above are presented in Table A.8.

Table A.8: Performances of simulations before, Figure 4.4 and after optimization, Figure A.13

Path Cost	
Before	After
350.99	349.09

TRITA -EECS-EX-2020:904