POLITECNICO DI MILANO
Scuola di Ingegneria Industriale e dell'Informazione
Master's degree in Automation and Control Engineering
Dipartimento di Elettronica, Informazione e Bioingegneria

PATH PLANNING OF COLLABORATIVE ROBOTS DEALING WITH REAL MAPS AND EXPLICIT TIME CONCERNS

Supervisor: Prof. Matteo Rossi
Assistant Supervisor: Ing. Claudio Menghi
Ing. Marcello M. Bersani
Prof. Patrizio Pelliccione

Master's thesis of:
Gabriele Belli, Matr. 853071

Academic Year 2016-2017

Alla mia famiglia

Sommario

Negli ultimi anni le applicazioni robotiche sono diventate parte integrante delle nostre vite. Piattaforme mobili e manipolatori vengono sempre più utilizzati nella nostra società per eseguire azioni ripetitive, alienanti o pericolose per un essere umano. I robot, fianco a fianco con i lavoratori, sono utilizzati in ogni settore industriale, dalla meccanica, chimica al settore manifatturiero, per il trasporto di materiali e persone, per le consegne e per l' E-commerce. Grazie alla loro flessibilità, personalizzabilità e vasta gamma di scelta nel mercato globale, questo genere di prodotto è in grado di soddisfare ogni tipo di richiesta e bisogno del cliente. Per questo motivo le aziende che si affidano ad applicazioni robotiche necessitano strumenti di programmazione sempre aggiornati in modo da poter gestire le loro richieste e bisogni.

Gli sviluppatori di software hanno il compito di garantire strumenti flessibili e riusabili in modo da compiere ogni necessità del consumatore. Il progetto Co4robots [38] all'interno del quale questa tesi è stata sviluppata, ha l'obiettivo di fornire strumenti che agevolano la programmazione nel contesto di applicationi multi-robot. Co4robots mira ad introdurre nel campo della robotica principi e tecniche caratteristiche dell'ingegneria del software per fornire un approccio ingegneristico alla programmazione di applicazioni robotiche. Il progetto analizza principalmente applicazioni multi-robot nelle quali un team di robot collabora al fine di raggiungere il soddisfacimento di una data missione, che risulterebbe non eseguibile da un singolo robot. Per esempio, una missione potrebbe richiedere al team di robot di caricare e muovere oggetti o rifornimenti in un determinato ambiente, dagli uffici, alberghi agli ospedali e sale operatorie. L' obiettivo finale assegnato al team di robot è descritto e assegnato come una missione definita da un linguaggio di alto livello.

Co4robots progetta di elaborare un approccio sistematico per lo sviluppo di applicazioni robotiche. Questo approccio include tecniche che riguardano diversi aspetti tra cui (i) la possibilità di lavorare con agenti eterogenei,

tra cui lavoratori e supervisori, (ii) lo sviluppo di algoritmi per la computazione delle azioni che i vari agenti devono eseguire per raggiungere la missione assegnata, (iii) un controllo decentralizzato che permette ai robot di interagire con l'ambiente circostante attraverso l'utilizzo di sensori e la comunicazione tra i componenti del team, (iv) facile integrazione con i diversi sistemi presenti sul mercato, (v) l'utilizzo di un linguaggio di alto livello per la definizione delle missioni, in modo da non richiedere all'utente particolari nozioni di robotica.

Questa tesi tratta uno dei vari aspetti considerati all'interno del progetto Co4robot: il planning. Il planning è un elemento fondamentale di ogni sistema robotico. Ha l'obiettivo di calcolare l'insieme di azioni e movimenti che il robot deve eseguire in modo da raggiungere l'obiettivo assegnato. Questo aspetto della robotica è già stato affrontato in letteratura, ma riteniamo che alcuni aspetti possano tuttavia essere approfonditi ed integrati. In particolare gli odierni strumenti per la pianificazione della traiettoria richiedono l'integrazione delle seguenti caratteristiche: (i) capacità di analizzare missioni che considerano aspetti temporali espliciti, per esempio missioni con vincoli espressi in secondi o minuti, (ii) abilità di gestire un team di robot composto da più agenti, (iii) considerare durante la pianificazione la possibilità di eseguire azioni collaborative che richiedono la sincronizzazione dei robot durante lo svolgimento della missione, (iv) considerare mappe realistiche che descrivono edifici ed ambienti reali. Al momento, in letteratura non è presente uno strumento che integra tutti questi aspetti in grado di supportare in modo efficace lo sviluppatore.

Questo lavoro propone un approccio che mira a risolvere i problemi descritti. Viene presentato uno strumento per la pianificazione della traiettoria che, per calcolare i comandi da assegnare ai robot, analizza (i) la mappa, in formato di immagine, dell'ambiente in cui gli agenti andranno a lavorare, (ii) il modello dei robot che descrive le relative caratteristiche di movimento e azione, (iii) la missione che i robot hanno il compito di eseguire considerando aspetti temporali espliciti. Queste informazioni sono utilizzati per generare un modello del sistema che verrà utilizzato nella fase di planning. Il modello del sistema è composto dal modello dell'ambiente e dai modelli dei robot che compongono il team, e viene descritto mediante l'utilizzo di Automi Temporizzati. La missione assegnata al team di robot viene invece specificata come una formula espressa in linguaggio TCTL (Timed Computational Tree Logic) che è in grado di gestire aspetti temporali espliciti.

Il modello descritto mediante gli Automi Temporizzati (che viene generato automaticamente) e la specifica TCTL vengono processate da Uppaal, un software che, analizzando tali input, è in grado di verificare l'esistenza

di una sequenza di stati e transizioni in grado di soddisfare la missione. In caso di risposta affermativa la traccia viene analizzata e le azioni vengono assegnate ed eseguite dai vari robot nel team.

La tesi è stata valutata sotto vari fronti. Abbiamo valutato il comportamento del planner considerando scenari realistici. Nei test eseguiti sono stati presi in considerazione 3 edifici, in particolare: "Jupiter Building", Chalmers University, Goteborg, Svezia, "Edificio20" ed "Edificio22", Politecnico di Milano. I piani di questi edifici sono state mappate utilizzando due diversi encoding e diverse tipologie di missioni sono state valuate. Per ogni tipologia abbiamo considerato vari bound temporali, il cui valore rappresentava il tempo massimo ento il quale la missione doveva essere soddisfatta. L'algoritmo è risultato capace di computare un insieme di azioni e movimenti rappresentati per mezzo di una traccia che, se eseguite dai robot, garantiscono il soddisfacimento della missione di interesse.

Per mostrare l'applicabilità dell'approccio in scenari realistici abbiamo prima di tutto valutato le tracce generati dal planner per mezzo del Choregraph simulator. Il simulatore ha dimostrato la possibilità di eseguire le tracce su robot reali, e in particolare sul NaoRobot. Abbiamo quindi valutato l'utilizzo dell'approccio su delle applicazioni realistiche. Abbiamo considerato il Turtlebot, una robot mobile in grado di muoversi in spazi brevi e stretti, e considerato tre diverse missioni nel quale il robot ha dovuto ritirare e consegnare il caffè, monitorare l'ingresso di personale non autorizzato ed eseguire il delivery di della merce. In tutti i casi l'approccio è risultato capace di (i) computare un piano per il robot partendo dalla mappa e da una specifica contenente vincoli temporale (ii) eseguire un forwarding corretto delle azioni e dei movimenti al robot. Abbiamo quindi valutato l'utilizzo dell'approccio all'interno di Pal Robotics [1], uno dei partner del progetto. Abbiamo considerato il TIAGO Robot, una piattaforma mobile di 145cm con un braccio robotico, e varie missioni provenienti da casi di studio reali. L'approccio è risultato efficace anche nel corso di questi esperimenti.

Abstract

In the last decades robotic applications pervaded human life. Mobile platforms, static manipulators, and mobile manipulators are increasingly used in our society to perform repetitive tasks. Robots side by side with human workers are deployed in every industrial field, from mechanic, chemical and manufacturing industries, to transports of goods and people, to smart delivery systems and E-commerce. Thanks to their high flexibility, customizability and the possibility to find on the market different products that are able to satisfy every customer's needs and requests, robots are selected for several purposes and employed in any possible working environment. For these reasons companies that rely on robotic applications for the fulfilment of their tasks need programming tools that must handle their demand. For this motivation software programmers should guarantee programmable, reusable, flexible tools in order to cover all the aspects of all the possible ideas and needs of the customers which absolutely is not a trivial responsibility.

The Co4robots project [38], within which this thesis has been developed, tries to overcome this issue. It aims at introducing software engineering principles and techniques within the robotic domain. The final goal is shifting towards well-defined engineering approaches which stimulate components reuse and have a final impact on the robotic market-places. Co4robots assumes a robotic application composed by a set of robots (i.e., a team) that aim at performing a set of missions in a collaborative way. A mission is an high level goal that a robotic application must achieve. For example, a mission may require a team of robots to bring medical supplies to a surgery room in the hospital environment. Those missions are typically defined in terms of a high-level complex mission - a formal description of the goals that robots shall achieve.

Rather than developing the components of the system from scratch each time a new robotic application is designed, or missions are changed, Co4robots tries to develop systematic software engineering techniques for the development of robotic applications. These techniques include among

others (i) new techniques that allow heterogeneous agents (including humans) to collaborate in manipulating and moving objects; (ii) decentralized real-time planning algorithms that allows synthesising the actions the robots in the team must perform; (iii) decentralized real-time perception to allow robots perceiving their environment either by using its own sensors or by exchanging sensory information with other nearby agents to accomplish specific tasks; (iv) a software integration platform to enable easy deployment of robots which includes configuration facilities. The platform aims at integrating the software produced within the project and drives the instantiation of the outcomes of research made into the project according to actual needs and (v) a user-friendly specification language that enables users (e.g., human users that utilize the robotic system in hand) without expertise in robotics and information technology (such as the personnel of a hotel or a hospital) to specify missions to be accomplished by the robots.

Path planning is a key point in robotic industry, planning aims to compute a set of actions and movements that a robot or a team composed by several robots are supposed to perform in order to fulfill a desired mission. Motion planning has already been studied and analysed in literature. However, we believe that some of the needs of current robotic are still to be handled. In particular current robotics intrinsically ask for planner tools that possess the following features: (i) analyze missions which contain explicit time concerns; (ii) manage teams of robots; (iii) consider during the planning procedure how robots are able to perform actions and synchronize among each other; and (iv) work on realistic maps. At the moment, in literature contributions a planner presenting all this features integrated in one single tool is missing.

This work proposes an approach that tries to solve all the problems described above. We present a motion planner that, in order to compute the commands to give to every agent of the team takes as inputs (i) the map of the environment in which the robots are deployed (in the format of images), (ii) the models of the robots that describe their possible movements, the actions they can perform, and how they synchronize, and (iii) a mission specification containing explicit time concerns. The approach takes as input a map describing the layout in which the robots will operate. A set of timed automata that describe the behavior of the robots. The mission the robot should achieve. The map of the environment is used to automatically generate a Timed Automaton (TA) that represents how robots can move within their environment. This TA is composed with other TAs that model how the robots behave. An high level mission describing the desired goal of the system (including explicit time concerns) is provided by the user as

a Timed Computational Tree logic formula (TCTL). The Timed Automata model and the TCTL specification are given to the program Uppaal Model Checker which creates a trace (a plan) that if performed by the robots ensures the satisfaction of the mission of interest.

This thesis has been evaluated under different aspects. We evaluated the proposing planning technique on realistic environments. In the tests performed we considered three real buildings, in particular: "Jupiter Building" of Chalmers University, Gothenburg, Sweden, "Edificio20" and "Edificio22" of Politecnico di Milano. The maps of this buildings have been transformed using two different encodings into a Timed Automata that describe the environment in which the robots are deployed. The two encodings have been evaluated considering several different missions. Each mission has been evaluated on different time intervals, which represent the maximum time available to execute the action.

We test the reliability of the approach both on the simulator and on real scenarios. Regarding the simulated experiments we evaluated the generated traces by the planner on Choregraph software simulator. The simulator showed positive results regarding the applicability on real case scenarios. The real experiments were conducted in the 'Jupiter Building" of Chalmers University and in the Pal Robotics Company (one of the partners of the project). In the experiments conducted in the 'Jupiter Building" of Chalmers University, we considered the TurtleBot robot, a mobile platform able to move in narrow environments. We ask the TurtleBot robot to perform different missions: bring the coffee to the offices, perform security surveillance during the night and delivery objects in the building. In all the cases the approach successfully achieved its missions, (i) compute a plan from the map and the specification, (ii) deploy the right sequence of commands to the robot. In the experiments conducted in the Pal Robotics Company, we deployed our planner on the TIAGO Robot, a mobile platform of 145cm with a robotic arm. We considered different missions and a set of real case scenarios. The approach succeeded in all the experimental tests performed.

Acknowledgements

I would like to express my gratitude to Prof. Matteo Rossi and Patrizio Pelliccione who gave me the opportunity to develop this thesis at Chalmers Univesity in Gothenburg, Sweden.

A special thanks goes to my supervisors, Claudio Menghi and Marcello Bersani, thank you for the huge help, the patience and the hard work, this project would not have been possible without your support.

I would like to thank my family, who have always encouraged and believed in me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides an overview on this thesis. Section 1.1 describes the research context. Section 1.2 presents the research problem addressed by this thesis. Section 1.3 provides a detailed description of the contribution. Finally, Section 1.4 describes the structure of the thesis.

1.1 Research Context

Robotic applications pervade human life. Mobile platforms, static manipulators, and mobile manipulators are increasingly used in our society to perform repetitive tasks. This occurs for example inside airports, where baggage management and delivery are mostly automatized by using appropriate robots, inside hospitals, where robots assist doctors during surgeries, and inside people houses, where autonomous vacuum cleaner are increasingly used to clean houses. Robots side by side with human workers are deployed in every industrial field, from mechanic, chemical and manufacturing industries, to transports of goods and people, to smart delivery systems and E-commerce. Thanks to new technologies and the growth of robotic and automation engineering this field has experienced a huge development on the global market. Future previews also show an increasing trend of development for this industrial sector. This spread of robotic applications has also been confirmed by the world Robotics Survey [2], which evidenced a boom on the use of service robots for professional use, and indoor logistics as a major robotic market. For example, sales of professional service robots had registered a growth of 28% in 2014 within the logistics segment. This resulted in an increase of USD 2.2 billion in the value of sales.

Alongside the adoption of robotic applications for industrial purposes and large scale production, the robotic world is slowly becoming an integral

and significant part of our everyday life. In our life we are starting to experience an increasing collaboration between human workers and robots in order to fulfil and ease daily tasks. Automated and programmed machines perfectly adapt to tasks and actions that for a human might result dangerous, difficult, repetitive, tedious and alienating. Adopting these kind of solutions, our working conditions and the quality of the product and service will significantly improve.

There exist several types of robots that differ one from the other for hardware components, structure, design, possibilities of movements, track-laying robots or with steering wheel, machines with completely different volume, from robots that measure few centimetres, to humanoid robots as tall as a person to even bigger robots. The most used types of robots in the industry nowadays are mobile platforms and static or mobile manipulators. Nevertheless all these differences they share the same characteristic of programmability and autonomous actions. Indeed, they are designed to perform actions under the supervision of a human worker and help and interact with him during the work.

Thanks to their high flexibility, customizability and the possibility to find on the market different products that are able to satisfy every customer's needs and requests, robots are selected for several purposes and employed in any possible working environment. For these reasons companies that rely on robotic applications for the fulfilment of their tasks need programming tools that must handle their demand. For this motivation software programmers should guarantee programmable, reusable, flexible tools in order to cover all the aspects and needs of the customers which is absolutely not a trivial responsibility.

Robotics developers are putting great effort in providing solutions that support effective robotic software development and the deployment of the produced software on real robots. These solutions try to expand the services that the robots could provide to the user, their ability of autonomous movements, path planning and actions in a determined environment, and to improve the collaboration and interaction with human workers. For these reasons the work of the programmers has a fundamental role in the development process, hence the software tools at their disposal must perfectly satisfy all their needs and requests.

Even if robotic developers are putting great effort in providing support to program robots, the production of software for robotic systems is still usually based on ad hoc solutions rather than being based on consolidated development processes and frameworks that support software development. This problem has also been evidenced by the H2020 Multi-Annual Robotics

Roadmap ICT-2016 [2] which states that usually, there are no consolidated system development processes for robotic applications. This is also highlighted by a lack of overall architectural models and methods.

The Co4robots project [38], within which this thesis has been developed, tries to overcome this issue. It aims at introducing software engineering principles and techniques within the robotic domain. The final goal is shifting towards well-defined engineering approaches which stimulate components reuse and have a final impact on the robotic market-places. Co4robots assumes a robotic application composed by a set of robots (i.e., a team) that aim at performing a set of missions in a collaborative way. A *mission* is an high level goal a robotic application must achieve. For example, a mission may require a team of robots to bring medical supplies to a surgery room in the hospital environment. Those missions are typically defined in terms of a high-level complex mission - a formal description of the goals that robots shall achieve. The capabilities (e.g., perception, manipulation, and actuation) of the robots with the team must be coordinated and supervised to ensure that the robotic application achieves the desired mission. For example, two robots can be employed to bring medical supplies to a surgery room in the hospital environment, i.e., a carrier that is able to bring the medical supplies to the surgery room and a manipulator which has the ability to load the medical supplies on the carrier. *Planners* for robotic applications are software components designed for this purpose, i.e., they synthesize actions that (a team of) robots must perform to achieve a given mission. For example, in order to ensure the satisfaction of the mission "bring medical supplies to a surgery room in the hospital environment" a planner may synthesize the actions "reach the hospital storage", "wait for medical supplies to be loaded" and "reach the surgery room" for the carrier and "wait for the carrier" and "load the carrier" for the manipulator. The actions computed by the planner are then forward on the running robots, which execute them.

Rather than developing the components of the system from scratch each time a new robotic application is designed, or when missions are changed, Co4robots tries to develop systematic software engineering techniques for the development of robotic applications. These techniques include among others (i) new techniques that allow heterogeneous agents (including humans) to collaborate in manipulating and moving objects; (ii) decentralized real-time planning algorithms that synthesize the actions the robots in the team must perform; (iii) decentralized real-time perception to allow robots to perceive their environment either by using its own sensors or by exchanging sensory information with other nearby agents to accomplish specific tasks; (iv) a software integration platform to enable easy deployment of robots

which includes configuration facilities. The platform aims at integrating the software produced within the project and drives the instantiation of the outcomes of research made into the project according to actual needs and (v) a user-friendly specification language that enables users (i.e., human users that utilize the robotic system in hand) without expertise in robotics and information technology (such as the personnel of a hotel or a hospital) to specify missions to be accomplished by the robots.

1.2 Research Problem

Within the context of the Co4robots project, one of the problems to address is to support developers in the creation of robotic applications. This is the general problem that is addressed by this thesis, by tackling the following sub-problems.

Problem 1. Developers of robotic applications are usually interested in missions that are specified in high-level logics that contain *explicit time concerns*, such as "a team of robots has to bring medical supplies to a surgery room in the hospital environment within 2 minutes". Explicit time concerns allow the specification of constraints in missions that not only predicate on the order in which events must occurs, but only on time aspects related with the mission execution, i.e., "within 2 minutes". Temporal logics have been largely used in the robotic domain to express robot missions. Linear Temporal Logic (LTL) has been extensively used to express missions [12], [14], [22]. However, LTL does not allow the specification of bounds on the delay between events, and therefore, a number of contributions have recently considered Metric Temporal Logic (MTL) [23] as specification language or Timed Computation Tree Logic (TCTL) [24]. Several works [14], [15], [23] in literature developed planners that considered missions specified using these logics considering the analysis of the movements over a bi-dimensional area, however, these planners are usually simple prototype planners that are only evaluated considering small matrices of cells that abstract small environmental maps. Scalability of the algorithms on real environments is rarely evaluated, as well as how the algorithms are able to address teams of robots when the size of the team increases. Furthermore, no automatic procedure is provided to generate the abstraction of the matrices that abstract the environment. Abstractions are usually created manually with the purpose of showing the applicability of the algorithm on simple scenarios. We believe that these limitations discourage the usage of such a planners in real applications. The developers are rather interested on using planners that work on real maps, and that can be used as off-the-self components.

Problem 2. Developers of robotic applications are usually interested in developing solutions that are able to manage *teams of robots*. Indeed, the majority of the robotic applications does not consider one single robot that performs a set of repetitive actions without any interaction with other robots. Even if planners for multiple robots exist, they usually do not compute the set of actions the robot must perform starting from a mission expressed in a logic language that allows the specification of explicit temporal concerns. Considering the planners proposed in literature that possess this feature, they usually not consider how robots can perform actions and synchronize among each other. They rather usually only take into account the collision avoidance constraints on small and simplified maps that abstract the environment without considering real cases or complex scenarios which is a fundamental requirement for the usage of the developed approaches in real scenarios.

Problem 3. Developers of robotic applications are usually interested in developing solutions that consider not only how robots move within an environment, but also which *actions they can perform* and how they *synchronize within each others*. By the term of synchronization we mean that two or more robots should be and remain in a determined position, usually one next to each other for an interval of time, where they are able to perform an action such as loading or unloading. A general purpose solution that allow considering movement, actions and synchronizations when missions contain explicit time concerns is still missing. Most of the existing solutions support LTL for the specification of the property of interest. The one that consider missions expressed in logics able to capture explicit time concerns do not usually consider teams of robots or are only evaluated on small and simplified maps that do not allow evaluating the applicability of the approaches in real case scenarios.

1.3 Contributions of the Thesis

The goal of this work is to develop an approach called MEMO (Map BasEd planner for TCTL MissiOns), supported by a tool, that helps the developers in the design of a robotic application. We focus on the planning problem. Specifically, we aim at computing the movements, actions and synchronizations the robots within a team must perform to achieve the satisfaction of a mission that contains explicit-time concerns. The proposed approach is comprehensive, meaning that it handles the problems identified in Section 1.2 from different perspectives and provides the following major features.

- **F1**: MEMO allows the user to express the desired mission in an high-level logic formula which may contain *explicit time* temporal constraints. For example, the user may express missions such as *"The robot should perform a determined action in a desired location within 60 seconds"*.

- **F2**: MEMO allows the management of *teams of robots*. For example, in order to ensure the satisfaction of the mission "bring medical supplies to a surgery room in the hospital environment within 60 sec" a planner may synthesize the actions "reach the hospital storage", "wait for medical supplies to be loaded" and "reach the surgery room" for the carrier and "wait for the carrier" and "load the carrier" for the manipulator and ensure that the execution of the computed plan does not require more than 60 secs.

- **F3**: MEMO allows considering not only how robots move, but also how robots can perform *actions* in the different locations together with the time required for executing actions. In the previous example, the action "load the carrier" performed by a manipulator may require 2 sec.

- **F4**: MEMO allows considering *synchronization* among robots. By the term of synchronization we mean that two or more robots should be and remain in a determined position, usually one next to each other for an interval of time, where they are able to perform an action such as loading or unloading. In the proposed example, the two agents must synchronize in the same location of the map and synchronously perform the actions "load the carrier" and "wait for medical supplies to be loaded".

- **F5**: MEMO works on real maps. The tool takes as input an image describing the layout of a building and a scale parameter and is able to compute plans based on the provided layouts. This feature makes MEMO simple to use in real case scenarios, even for non expert users.

Our solution, MEMO allows providing the features previously described as follows.

- MEMO takes as input an image describing the layout of a building (F5) and generates a timed automata that encodes this layout and show how a robot can move within its environment. We implemented two possible encodings to describe the environment and its relative

spacial constraints. The first one is more graphical while the second one relies on logic formulas to describe the ambient. The generation of the timed automata is based on a sampling step that specifies how the state space of the building must be discretized. The user is supposed to set few parameters as input in order to specify the level of details reported on the Timed Automata model from the map on the image. One of these parameter is called "sampling span" given by the user in meters. This value represents the distance between two consecutive considered points in the environment. The algorithm that manages the creation of the model from the image creates a matrix that follows the proportions of the map and organizes the elements at a distance specified in the inputs. The points of the matrix are defined with an identifier, and thanks to this identifier combined with the "sampling span" it is possible to create a direct connection between the points of the matrix with the real points of the environment. In order to describe the obstacles present in the environment, the algorithm parses all the pairs of consecutive states, considered in all directions and orientation, and checks if between the correspondent points in the real environment there is an obstacle or the path between them is clear. Then the procedure reports these information in a model expressed using a Timed Automaton that encodes all the relevant characteristics of the environment, including the walls, rooms, doors and the corridors.

- MEMO models the behavior of the robots by means of a Timed Automaton. Nowadays in the world of robotics there exist a large variety of robot models, each one with its own properties of movements and different behaviors, in order to fulfill his missions the user selects the best one relative to his goal. In order to deliver a tool capable of covering a large number of situations and robot configurations MEMO allows the user to customize and personalize the Timed Automaton model of the robot. This feature guarantees a good description of the real robot selected by the user. In our simulations and verifications we selected a general robot able to move in the space in four different directions perpendicular to the vertical and horizontal axis of the map and able to perform one or more actions in a determined point of the map. The model is divided in two parts, the first one describing how the robot moves in the environment while the second one manages the actions and the task that the robot is able to perform in the space.

  In the considered scenario the first part of our model is composed by five states, four of them representing the direction of movements "up",

"down", "left", "right" and the fifth one representing the configuration in which the robot is stationary. The way in which these states are linked together defines different configurations of robots and different ways of movements. Each transition is labelled with the action that represents, which is the name of the target state of the transition. For example, if we are in the configuration "stay" and the machine performs the action "right", after a defined amount of time we will end up in the state "right". The transitions are fundamental to manage to time necessary to achieve a mission, indeed considering a local clock on the robot, which is reset to zero after the execution of every action, the algorithm sets constraints on the time interval before enabling the following action. This time bound depends on the velocity of the robot and the distance that the robot should travel.

The second part of the robot model describes and programs the execution of actions through an additional set of states. The transitions leading to these states constraint the position of the robot in the environment, i.e., they specify that certain actions can only be performed in specific positions. Additional constraints are also added when sequences of actions must be performed in order.

- MEMO allows dealing with *explicit time* temporal specifications and concerns. We considered Timed Computational Tree Logic (TCTL) for the specification of the mission of interest. Using this logic, MEMO is capable of answering requests such as "Is the robot able to deliver within 2 minutes?" or "Is the robot able to visit three different rooms in a specific order within 10 minutes?".

- MEMO relies on UPPAAL Model Checker. UPPAAL is an integrated tool for creating models, validation and verification of real-time systems implemented as networks of timed automata, extended with data types. UPPAAL supports the verification of specifications written in TCTL language, which allows the user to describe the behaviours that a robot may perform. In MEMO, UPPAAL is used to analyze the model of the system, obtained by combining the model of the robots and their environment, and the TCTL mission, and is used to produce the plans that the robots must perform. A key point of exploiting UPPAAL program is the opportunity to extract traces after the verifications that represent the motion plan of a robot. The program taking as input the explicit time dependent specification expressing the mission that the robot should accomplish, analyses the Timed Automata

model and, if the proposed structure is able to achieve the mission, the outcome of the tool will be a trace representing the organized sequence of all the states of the model from the initial to the final configuration. From this sequence of states we manage to extract the practical motion plan. Using UPPAAL allows us to go beyond the limit of the proof-of-concepts solutions.

MEMO has been evaluated on different perspectives. Specifically,

- We evaluated the planner considering maps coming from *real case scenarios*. Our maps represent layouts of real buildings. We selected the maps of the Jupiter Building of "Chalmers University", Gothenburg, Sweden, Edificio20 - DEIB of "Politecnico di Milano", Edificio22 of "Politecnico di Milano". These maps represent examples of reasonable size real environments. The approach and the planner framework have been validated with different experiments, considering decreasing values for the sampling, different types of robots and different values for the secondary parameters. We also considered different missions and we verified them with different time boundaries.

- We considered both a single robot moving in the space and performing tasks and a team of robots composed by two interacting agents programmed for performing a global common mission.

- We evaluated the planners on a *simulated environment*. We performed some experiments on the robotic simulator Choregraph which is a platform for the Nao Robot [3]. Nao is an autonomous, programmable humanoid robot developed by Aldebaran Robotics. Thanks to this simulator we were able to simulate the Nao robot performing different missions.

- We evaluated the planner with *an experimental evaluation*. We deployed our tool on real robots in real environments. We performed the experiment on the Tiago robot which was provided by Pal Robotics, one of the partners of the Co4robots project. The Tiago robot is a mobile manipulator 1.45 meters tall, with 7 degrees of freedom able to move in complex environments and perform actions. We used MEMO to compute a plan that satisfy a given mission and to forward the generated plans to the Tiago robot that executes it. We also performed experiments with a smaller moving platform called TurtleBot. We considered missions in which the robot had to reach a set of points

of interest within a real environment, and perform a set of actions in these points.

1.4 Structure of the Thesis

This thesis is organised as follows:

Chapter 2 - State of Art.
It analyses the related work. We considered contributions that handled models based on Timed Automata. We studied works that handled explicit time concerns, looking for work capable, as the one proposed in this thesis, to deal with time bounds expressed in second, minutes or hours. We evaluated the logic languages supported by planners proposed in literature. We also considered the capability of the tools to manage a team of robots moving and performing actions.

Chapter 3 - Background.
It describes background concepts and notation necessary to understand this work. We describe Timed Automata (TA) and Temporal Computational Tree Logic: the languages adopted in this work to describe the model of the system (robots and their environment) and the missions of interest. We briefly introduce the UPPAAL Model Checker, explaining the support provided by this tool to generate of a TA model, to simulate its behavior, and to verify the properties of interest. Finally we introduce the material used for the practical experimentation: the software exploited for the simulation, and the robots used for the experimental evaluation (Tiago Robot of PalRobotics Company and the TurtleBot).

Chapter 4 - Contribution.
It presents the contribution of this thesis. We describe the proposed tool: MEMO a Map basEd planner for TCTL MissiOns. MEMO is a planner framework able to compute the motion plan of a robot or a team of robot taking as input an image of the environment and a mission specified in a specific time dependent logic. We present the work flow of the tool and its main characteristics. We describe two encodings used to transform a map of the environment into an equivalent timed automaton. We show how robots are modelled using timed automata and how these models are used to compute the actions that that the agents must perform.

Chapter 5 - Implementation.
We describe how MEMO works in practice. We discuss how the Automata system is created and saved. We describe the algorithms implemented in MEMO, how TA are extracted from an image. We explain the scripts and

the functions to make use of our tool. We present how we performed the translation from the UPPAAL traces to the commands given to the robots.

Chapter 6 - Evaluation.
We report on the experiments performed to evaluate MEMO and on the analysis of the results of the experiments. We describe how we generate the different inputs considered in the evaluation procedure: the maps, the robots in the robotic application, and the missions they have to perform. We evaluate the time required by MEMO to compute plans in each experiment. We evaluate the results and used MEMO in different real scenarios.

Chapter 7 - Conclusions.
We draw conclusions and present future works.

Chapter 2

State of Art

Section 2.1 analyses the contributions present in literature related to our thesis. Section 2.2 classifies the related work considering the problems addressed in this thesis, to detect if and how these problems have been addressed in literature.

2.1 Related work

We analysed contributions presented in literature regarding the topic of path planning for robotic applications, when the mission of interest was specified through an high level logical language. We considered both the single and the multiple robots cases, whether the approaches support synchronization and collaboration of robots for the mission achievement, and whether real environments were considered.

Andersen et al. [8] present a solution to the motion planning problem for multiple mobile robots. They described how a network of interacting timed automata could be used to define a model, analyse and verify the motion planning problems for systems with multiple robots system that perform displacement within the same environment. Fundamental to multi-robot problems is the need to control and plan the motion such that the robots do not collide with each other or obstacles. This paper focuses on the motion planning problem for a group of unicycle type robots. It provides a framework for establishing plans based on synchronized timed automata. The proposed model takes into account a team of robots moving in a small planar grid in which the motion of each agent is limited by the layout of the workspace. The timed automata model describing the system represents an abstraction of the possible motion and of the planar grid representing the environment. Considering this abstraction, the paper present a procedure

that allows composition and formal symbolic reasoning about plans. The contributions relies on UPPAAL program in order to check the specifications and perform verifications on questions such as:

- "Will the robots collide?"
- "Are the robots able to to reach their goals?"

In this way it is possible to check and verify safety and liveness questions. Dealing with the software UPPAAL the specification requirements are formulated in computational tree logic (CTL). The output of this approach are plans that are subsequently used as a high level motion plan and transferred to the to the robots' control system. The presented approach has been evaluated on three different tests:

- Deadlock test
- Motion planning test - ability to find a feasible path
- Motion planning test - ability to report an error

The results carried out by this paper show an expensive computational cost even if abstraction and approximation have been introduced in the model and considering the scalability of the approach it will imply an exponential growth of the complexity.

Chen et al. [10] present an algorithmic framework in order to manage a team of agents considering a task specification given as an LTL formula concerning a set of properties. Taking into account agents able to satisfy the desired missions and cooperate with the other agents, this work proposes individual control and communication strategies in order to achieve a global behaviour of the system that satisfies the given specification. This contribution considers a purely discrete scenario, in which the behaviour of the agents is described as a finite transition system and finally applies the method to automatically deploy a team of miniature cars in an urban-like environment as a practical experiment.

This approach deals with the challenge of constructing a finite models that accurately capture behaviours of dynamical system. This paper addresses a discrete problem in which the agent is modelled as a finite transition system.

The method proposed takes as inputs three main elements:

- Set of properties to be satisfied

- Team of agents capable of dealing with the desired missions, cooperate with each other.
- Task specification in the form of LTL formula explaining the temporal and logic constraints of the mission.

The proposed approach aims at finding out individual and control and communication strategies for each agent such that the task is accomplished. Considering this goal, the main contributions of this paper might be analysed under three different aspects:

- Development of a computational framework to synthesize individual control and communication strategies from global specifications given as LTL formulas over a set of interesting properties.
- Extension of the approach of checking closure properties of temporal logic specifications in order to generate distributed control strategies for a team of agents considering their dynamics.
- Practical experimentation on a set-up composed by robotic cars automatically deployed from specifications given as LTL formulas.

In a successive work, Chen et al. [11] develop a technique able to automatically generate a control policy for a single robot moving in an environment that presents some elements with partially unknown or changing behaviour. The agent was supposed to achieve an optimal surveillance mission, in which the request defined by the user need to be executed and repeated several times while the expected time in between consecutive services is minimized. The main idea of this work is to define a fragment of Linear Temporal logic (LTL) to describe the desired mission and analyse the problem as a temporal logic game. The two main aspects treated in the contributions are:

- Extension of the results in automata learning to detects patterns of partially unknown behaviour
- Employment of an automata-theoretic method in order to generate the control policy.

This paper considered Linear Temporal logic (LTL) and aimed at bringing together automata learning methods from the fields of theoretical linguistics and techniques from temporal logic games in order to develop a control strategy for an agent that moves in an environment that might present unknown dynamics. The main contributions of this paper might be summarised in two points:

- Specification of a fragment of LTL to describe the tasks that the agent is supposed to perform.

- Show that under some assumptions the environment dynamics could be seen as a subclass of w-regular languages called strictly k-local languages.

Fainekos et al. [12] present an automatic framework for the solution of temporal logic motion planning problem for dynamic mobile robots. The presented approach is based on hierarchical control the notion of approximate simulation relations and a new definition of robustness for temporal logic formulas. As output of analysis and implementing this procedure two main results have been obtained:

- Presentation of an automatic framework for the solution of the temporal logic motion planning problem for kinematic models.

- Construction of a robust solution accounting for bounded errors in the trajectories of the system.

This work also deals with with the problem that arises in path planner unable to manage a number of goals or actions that must be executed in a particular order or without considering temporal extended missions.

Guo et al. [14] present a reconfiguration method for the motion planning of multi-agent systems under unfeasible local LTL specifications. This work shows algorithms in order to compute optimal plan candidates classified depending on the implementation costs and their distance from the specification. The main idea is based on the relaxation of not feasible specification. The main contribution is the introduction of a metric within the atomic proposition domain, and the relative weighting between the implementation cost of a motion plan and its distance from the original specification.

In successive works Guo et al. [15] present a systematic way to synthesize a hybrid control strategy for motion and action planning of an autonomous robot under LTL task specifications. It proposes a new framework that combines model-checking-based robot motion planning that deals with action using action description languages by means of Linear Temporal Logic (LTL) formulas. An optimal planner is then designed in order to generate a sequence of motion-and-action plan able to achieve the desired mission specified by the user in the requested temporal logic. This contributions deals with the problem of those approaches that carry out the motion planning and action planning independently since the actions and the plan are closely

related. It follows the idea that the specifications of "where to go" is strictly correlated by the specification of "what to do" both in current and future instants. Therefore the final goal is to incorporate the action description formalism with the motion planning. Summarising the main contribution of this paper, it proposes generic framework to derive the complete description of the robot's functionalities within a certain workspace, such that any LTL task specification in terms of desired motions and actions can be treated. Furthermore, an optimal sequence of robot's motion and action that satisfies the given task specification.

Guo et al. [17] also propose a distributed motion and task control framework for multi-agent systems under complex local LTL tasks and connectivity constraints. Thanks to this planning, the team of robots is supposed to achieve all the individual and collaborative tasks while at the same time connectivity constraints are satisfied. One of the main points presented is the fact that each local LTL task specification captures both the requirements on the respective agents behaviour and the requests for the other agents collaborations needed to accomplish the task. The presented solution shows a decentralized coordination between the robots concerning a leader-following scheme, that combined with a systematic leader change, ensures that all the task will be accomplished. Going a bit more into details, this paper focuses on planning under complex tasks assigned to the agents such as periodic surveillance, sequencing or request-response. Agents are considered as a team modelled as a dynamical system with an assigned local task specification expressed in Linear Temporal Logic (LTL). The aim of the work is to find a suitable decentralized solution while taking into account the constraints that the agents can exchange messages only if they are close enough. The contribution of this work might be summarised in three main points:

- the continuous controller is distributed and integrated with the leader election scheme;
- the distributed leader election algorithm only requires local communications and guarantees sequential progresses towards individual desired tasks
- the proposed coordination scheme operates in real-time, upon the run of the system as opposed to off-line solutions that require fully synchronized motions of all agents.

Guo et al. [18] present a distributed hybrid control strategy for multi-agent systems where each agent of the team has a local task specified as a Linear Temporal Logic (LTL) formula and at the same time is subject

Kloetzer et al. [22] present a computational procedure that enables the automatic synthesis of decentralised communication control strategies for a robotic team from global specifications, where these specifications are assigned as temporal and logic statements and deal with the missions of visiting determined regions of interests in a partitioned environment. On this work simulations and experiments have been done. This paper is motivated by the idea of develop a path planner able to specify motion task in a rich, high level language and then have this specification automatically converted into a set of low level primitives, such as feedback controllers and communication protocols, in order to accomplish the desired task. This contributions tries to find a solution for two main problems in current motion planning in robotic applications.

- In most of the cases the path planning is limited to answer questions such as "go from the initial point A to the final point B while avoiding collisions with obstacles and other robots" which is nor rich enough to describe large and complete missions which on the other hand a robot or a team of robots are able to perform.

- The second aspect deals with the unanswered question "Is a robotic application able to automatically generate provably correct local communication and control strategies from global missions specified by the user in a rich, complete and natural language over regions of interest considering a real environment?"

This work considers a discrete problem, in which a team of agents can move through the vertices of a graph representing the environment. The robot communication constraints are implemented as a graph while the motion system is modelled as a transitions system over the graph. The control strategies are automatically generated from task specifications given as a Linear Temporal Logic (LTL) formula. Analysing the results,the approach proposed showed five main disadvantages:

- Very high computational cost and the possibility to work only with very limited environments and with teams composed by a small number of agents.

- The solution is not completely decentralized, the solution is found in a "centralized" manner and executed in a distributed manner.

- The approach is conservative or incomplete, it might fail or might be unable to find an existing solution.

- It cannot deal with changes in the environments or external events.
- The method captures no measurements and control uncertainty.

John Koo et al. [23] propose a framework for the coordination of a network of robots with respect to formal requirement specifications expressed in temporal logics. A regular tessellation is used to partition the space of interest into a union of disjoint regular and equal cells with finite facets, and each cell can only be occupied by a robot or an obstacle. Each robot is assumed to be equipped with a finite collection of continuous-time non-linear closed-loop dynamics to be operated in. The robot is then modelled as an automaton for capturing the modes of operation for either staying within the current cell or reaching an adjacent cell through the corresponding facet. By taking the motion capabilities into account, a bi-similar discrete abstraction of the hybrid automaton can be constructed. Having the two systems bi-similar, all properties that are expressible in temporal logics such as Linear-time Temporal Logic, Computation Tree Logic can be preserved. Motion planning can then be performed at a discrete level by considering the parallel composition of discrete abstractions of the robots with a requirement specification given in a suitable temporal logic. The bi-similarity ensures that the discrete planning solutions are executable by the robots. For demonstration purpose, a finite automaton is used as the abstraction and the requirement specification is expressed in Computation Tree Logic.

Koymans et al. [24] aim to present a formal specification method for real-time systems concerning the main role of quantitative temporal properties. This work characterises real-time systems by giving a classification of such quantitative temporal properties, then it extends the usual models for temporal logic by including a distance function to measure time and analyses what conditions should be applied on such function. The introduction of appropriate temporal operators turns qualitative aspects into quantitative ones.

Nikou et al. [33] present a systematic method for multi-agent controller synthesis which aims to compute cooperative plans under high-level specifications given in MITL formulas. The proposed solution involves a sequence of algorithmic automata in order to compute the desired mission. The goal of this work is to introduce some specific time bounds into complex tasks, such as:

- Periodically survey of a region
- Avoiding a particular location in the space

- Keeping the interval between the execution of two consecutive actions smaller that a certain value of time instants.
- Visiting some location with a determined interval of time.

The structure of the contribution in order to address the problems is:

- The robot dynamics are abstracted into a finite discrete transition system using cell decomposition.
- A discrete plan that meets the high level mission is synthesised.
- The discrete plan is translated into a sequence of continuous controllers for the original system.

This work aims at designing an automated planning procedure for a team of agents. The user gives individual, independent timed temporal specification to each robot and also global team specifications. The considered logic is Metric Interval Temporal Logic (MITL) in order to specify explicit time constraints. The solution proposed is decentralised in handling the individual missions and centralized only when dealing the global team specifications.

Quottrup et al. [35] present how a network of interacting timed automata could be exploited to model, analyse and verify the motion planning of a team of multiple robots able to move in a determined environment. The proposed method relies on on an infra-structure of agents with feed-back controllers that presents constraints on the movements defined by a planar grid. The automata language allows the composition and formal symbolic reasoning about coordinates solutions even if the environment considered is just an abstraction of a real ambient. The mission given to the agents are specified in Computational Tree Logic (CTL), and relying on the software UPPAAL, it is possible to verify these properties. In this way it is possible to algorithmically analyse all feasible trajectories that satisfy the desired missions.

Rabiah et al. [36] focus of navigation algorithms and path planners, which are two fundamental features of autonomous robots, aiming to produce some improvements. The goal of this work is to capture concrete specifications by transforming a hing-level specification into an equivalent executable program. This paper uses Z specification method in order to define the motion that an agent is supposed to perform.

Smith et al. [37] present a method for the automatic generation of optimal path for a moving robot, able to satisfy specifications defined in an high level language. The environment and the motion of the robot is represented

as a general transition system characterised by transitions. The desired behaviour of the robot is specified as a linear temporal logic formula, thanks to LTL language. This work also focuses on the optimization of a proposition which must be repeatedly satisfied. The cost function that should be minimised is the time between instances of the optimizing proposition. A trajectory is computed for every formula, and this trajectory minimises the function. This method utilises Buchi automata to produce an automaton, structured as a graph, able to satisfy the properties representing the missions.

Work on cyber-physical spaces [46, 43, 34] has targeted modelling and obtaining automata structures for cyber-physical spaces in an automatic fashion supporting their verification. Moreover, planning –over different runtime assumptions– has been considered for adaptive security, targetting explicitly the interplay between computational and physical aspects [45]. We note that analyzable models of the physical space where robots operate may be automatically obtained [42, 44] and model-driven engineering principles and standards can integrate them in our approach [19].

2.2 Classification of related work

The goal of a motion planner is to compute a set of actions, a plan, that a robot or a team of robot should perform in order to achieve a predetermined mission or task specified by the user. In this thesis we present MEMO, a motion planner, which aims to become an useful tool in the development of a robotic application. As presented in the previous section, the topic of motion planning has been studied and considered in different works in literature, we believe that some aspects and needs of current robotic applications are still to be handled, improved or integrated in order to reach a user-friendly and complete approach, which is one of the goals proposed by this work. For this reason we classified work in literature considering four different aspects:

- the ability of the proposed planning framework to support missions that contain explicit time concerns;
- the ability of the proposed planning framework in managing teams of robots;
- the ability of the proposed planning frameowrk in managing actions and synchronization among robots;
- the ability of the proposed framework in supporting realistic environments.

	F1		F2	F3		F4		
Paper	Log	Tp	Teams	Act	Sync	Mp	Exp	Tool
[35]	TCTL	✓	✓	✗	✗	✗	✗	UPPAAL
[8]	TCTL	✗	✓	✗	✗	✗	✓	UPPAAL
[23]	CTL	✗	✓	✗	✗	✗	✓	C-SMV
[41]	LTL	✗	✗	✗	✗	✗	✓	✗
[37]	LTL	✗	✗	✓	✗	✗	✓	✗
[17]	LTL	✗	✓	✓	✓	✗	✗	Matlab
[20]	LTL	✗	✓	✗	✗	✗	✓	✗
[10]	LTL	✗	✓	✓	✓	✗	✓	✗
[21]	LTL	✗	✓	✓	✓	✗	✗	✗
[22]	LTL	✗	✓	✓	✓	✓	✓	✗
[12]	LTL	✗	✓	✓	✓	✓	✗	Matlab
[40]	LTL	✗	✓	✓	✓	✗	✗	Matlab
[11]	LTL	✗	✗	✓	✗	✗	✗	Matlab
[39]	LTL	✗	✓	✗	✗	✗	✗	✗
[16]	LTL	✗	✗	✗	✗	✗	✗	✗
[14]	LTL	✗	✓	✓	✓	✗	✗	Matlab
[15]	LTL	✗	✗	✓	✗	✗	✗	Matlab
[18]	LTL	✗	✓	✓	✓	✗	✗	✗
[33]	MITL	✓	✓	✓	✓	✗	✓	✗
[47]	MITL	✓	✗	✓	✓	✗	✗	CPLEX
[32]	MITL	✓	✓	✓	✓	✗	✗	Matlab

Table 2.1: Analysis of related work on planning.

Table 2.1 presents the obtained results, where

- Log: reports the logic language that has been used to specify missions;
- Tp: we show which works deal with explicit temporal concerns;
- Team: in this column we analyse if the planner presented in the relative work could manage more than two robots moving at the same time in the same environment;
- Act: shows if the procedure under study may be able to allow the user to program and personalize the system with different robot actions with relative different time bounds;
- Synch: shows whether synchronization has been considered;

- Mp: reports the works that take into account a map of a real and complex environment, such as an entire floor of a building, not only considering an abstraction or simplification of the space;
- Exp: in this column we list the contributions that performed real experiments and deployed their algorithms on a real robot;
- Tool: in the last columns we describe with which software tool each work has been developed.

Note that, when dealing with multiple robots, with the term synchronization we mean their ability to perform collaborative actions, in order to achieve this the robots must be able to meet in the same location at the same time and perform an action together, for example a collaborative grasping.

In the following we discuss the limitations of the work presented in Table 2.1, and the absence of a generic framework that handles all the aspects previously described.

2.2.1 Expressing explicit time concerns in robotic missions

In literature the logic language that has extensively been used to express missions is the Linear Temporal Logic (LTL). It is a modal temporal logic with modalities referring to time. In LTL, one can encode formulae about the future of paths, e.g. a condition will eventually be true, a condition will be true until another fact becomes true. It is a fragment of the more complex CTL, which additionally allows branching time and quantifiers. This type of logic turns out to be enough expressive to specify a rich variety of possible behaviours of the robots [12], [14], [22], different missions and combinations of actions and also benefits from a consolidated knowledge containing algorithms for verifications, model-checking and synthesis of controllers.

However, this kind of logic applied to a robotic applications shows some drawbacks, in fact the lack of metrics on time does not allow the user to define specifications of bounds on the delay between events. In LTL the programmer could express constraints on the order in which events should occur for example it is possible to request that an event B follows an event A but the user is not able to express limitations on the time interval between the two events by imposing for instance a delay smaller than 20 time units. For this reason, in recent works, some contributions [24], [8], [23], tried to take into account a different logic language which considers more detailed time constraints, the introduced language are Metric Temporal Logic (MTL) or Timed Computational Tree Logic (TCTL). Even if these works adopt this kind of temporal logic, the majority of them are mostly theoretical, some

of them do not explicit consider how the robots synchronize one with the other and there is not the possibility to program the actions of the robots, while in our work all this feature are integrated in the framework. Another huge difference between the tool presented in this thesis and the previous ones adopting the same temporal logic is that they are not validated on real maps as they consider small abstractions of the environments. Considering this aspect we propose a more complete and exhaustive approach.

In the second column we list the works that consider explicit time concerns for mission definition, the ones that are able to deal with specifications such as "*Is the robot able to perform action A within 60 seconds starting from the initial configuration?*". As shown in the table most of the contributions do not deal with with timing specifications, and in some cases (such as [24]) the missions are specified without taking into account explicit time constraints even if the logic language adopted for the work supports it. In [33], for example the language used do not allow the user to define specific time limitations, the tool simply runs an algorithm that computes the plan with shortest path.

2.2.2 Managing Team of robots

In modern robotics the majority of the automated work are performed by the collaboration of robots, working together they are capable of achieving difficult and complex tasks. For this reason, in a motion planner software it is fundamental to give the opportunity to the user to manage a team of robots. The analysis of the behaviour of a team of robots and the synthesis of the motion plans that regulate their movements over a bi-dimensional area have been studied in several works in the last few years [14, 36, 23, 32].

In modern robotics studying this kind of features means considering two or more heterogeneous agents that may be have different structure or hardware but share the same properties of programmability and ability to read specifications given as input in order to perform collaborative actions. Even if these consideration have been taken into account in several works in most cases the explicit temporal constraints on which we are highly focus are not considered.

Concerning the problem of managing multiple robots in the same environment, another improvement of our work with respect to the previous contributions is the support of the Collision Avoidance algorithm also on large maps. This aspect is a fundamental feature, it is not possible to work with a planner that do not consider the possible conflict of two robots in the same position. Unlike the analysed works we are able to deal with a

multiple robots system also in a extended environment.

2.2.3 Handling Actions and Synchronization

In Table 2.1, columns *Act* and *Synch* describe works that consider how robots can perform actions and synchronize among each others, respectively. As shown in the table not all the works include this feature. The one that possess this feature, generally do not consider explicit time concerns. We think that this aspect is rather foundamental and must be considered in our approach.

2.2.4 Considering realistic Environments

Most of the planners studied in literature are not suitable for a practical use since they cannot support simulations, verifications and path planning on extended and real case scenarios. The majority of the contributions listed in the Table 2.1 consider just an abstraction of real environments. They do represent the space through grid cells, usually with very limited size. In these works the environment took in consideration is composed by one room or one ambient where few obstacles are introduced directly by the user just considering an ad-hoc solution. With this kind of procedure would be impossible to model larger ambient with a greater number of obstacles, doors or corridors normally present in the environments where robots are deployed.

For these reasons it is difficult to evaluate how the algorithms proposed scale when the size and the complexity of the maps grow, for example in the case of applying the planner to real buildings. Moreover, most of the planners are developed by means of a particular and specific solutions rather than built on pre-existing solutions of proved effectiveness. Our tool presents a solution to this problem as it supports large maps as input and thanks to the developed algorithms we are able to create a model of a great variety of ambient and real buildings.

The use of consolidated tools has many advantages. In many cases the implemented procedure is stable and efficient as the tools include optimizations that work at the engine level. Moreover, consolidated tools might offer different options to the user for the analysis of the system, such as, for instance, the state space exploration policy (either breadth first or depth first). As as consequence, the user can experiment the most appropriate means to carry out the analysis by choosing different available options without the need of implementing specific solutions on every considered system.

Over all the past contributions analysed on the grid, only two of them [24], [8], rely on UPPAAL, which is a well known model checker for time dependent systems. However these works do not allow the user to define complex mission specifications concerning actions and synchronization among robots which are considered a crucial feature of every motion planner framework. Further to this these works carry out an evaluation on a synthesized spaces of small size without the application to a real scenario.

Finally the last columns of the table shows if a contributions has been validated and implemented on a real robot moving in a real scenario. With the exception of [24], [8], the remaining works have only been evaluated through simulations which sometimes does not perfectly reproduce all the characteristic of the real world. For this reason after the verification, simulation and evaluation of our tool on a software simulator we have also tested its validity performing real experiments on actual robots.

Chapter 3

Background

This chapter describes the concepts and formalisms necessary to fully understand the contribution of this thesis. Section 3.1 provides an overview over the notation of Timed Automata and of the temporal logic language adopted in this work. Section 3.2 presents the UPPAAL Model Checker: the main software on which our planning procedure is based. Section 3.3 shows the tools that will be used during the evaluation.

3.1 Modeling Formalisms

Section 3.1.1 presents Timed Automata. Section 3.1.2 presents the Timed Computational Tree Logic, i.e., the logic that will be used in the specification of the missions of interest.

3.1.1 Timed Automata

A finite-state machine is a mathematical model of computation. It is an abstract machine that is located in just one of the finite number of states that represent the system, at any given time. Considering and receiving external inputs the described system changes from one state to another, this change is made possible by transitions that put in relation a pair of states, enabling the system to modify its configuration. A finite-state machine is defined by the list of the states composing the system, the initial configuration and the conditions applied on the transitions in order to enable or disable the change of configuration.

The behaviour of state machines can be observed in many devices in modern society that perform a predetermined sequence of actions depending on a sequence of events with which they are presented.

In theory a Timed Automata [7] is a finite automaton extended with a finite set of real-valued clocks. During the development of the behaviour of the system and the continuous changes of the configurations clock values increase all with the same speed. Considering the transition between one state and another, clock values could be compared to integers. This feature allows the definition of conditions on the transitions concerning the values of the clocks, which are able to enable or disable the change of configuration. Furthermore, in order to give flexibility to the program these clock could be reset to zero in order to restart the count of the time.

Let X be a finite set of *clocks* with real values, Y be a finite set of *variables* with integer values and Act be a set of actions. Let $\Gamma(X)$ be the set of *clock constraints* defined as $\eta := x \sim c \mid \neg\eta \mid \eta \wedge \eta$, where $\sim \in \{<, =\}$, $x \in X$ and c is a natural number; and let $\Gamma(Y)$ be the set of *variable constraints* ζ defined as $\zeta := y \sim d \mid y \sim y' \mid \neg\zeta \mid \zeta \wedge \zeta$, where y and y' are variables in Y and d is an integer number. Let $assign(Y)$ be the set of assignments of the form $y := exp$, where $y \in Y$ and exp is an arithmetic expression over the variables in Y and the integers.

Definition 3.1.1. Given a set of atomic propositions AP, a *timed automaton* is a tuple $\langle Q, q_0, v^0, I, L, T\rangle$, where: Q is a finite set of locations, $q_0 \in Q$ is the initial location, v^0 is a function assigning each variable in Y with an integer value, $I : Q \to \Gamma(X)$ is an invariant assignment function, $L : Q \to \wp(AP)$ is the labeling function and $T \subseteq Q \times Q \times \Gamma(X) \times \Gamma(Y) \times Sync \times \wp(X) \times \wp(assign(Y))$ is a finite set of transitions such that $Sync = Act \times \{!, ?\}$.

The semantics of a TA is given in terms of *configurations*, i.e., pairs (q, v) defining the current location of the automaton and the value of all clocks and variables, where $q \in Q$ and v is a function over $X \cup Y$ assigning every clock with a real and every variable with an integer. A configuration change from (q, v) to (q', v') can happen because either a transition in T is taken or because time elapses.

3.1.2 Timed Computational Tree Logic

UPPAAL properties allow for specifying missions through a BNF-grammar based on the CTL logic, which also contains "time related" constraints (TCTL). Let e be a boolean combination of formulae on variables and clocks such as, for instance, $x \leq 10 \wedge c_1.Done$ indicating that clock x is less than or equal to 10 and that component c_1 is in location $Done$. TCTL allows the specification of properties in the form

- $A\Box e$ ("for all path globally e holds");

- $A\Diamond e$ ("for all paths finally e holds");
- $E\Box e$ ("exists a path in which always e holds") and
- $E\Diamond e$ ("exists a path in which finally e holds").

3.2 UPPAAL Model Checker

UPPAAL [25] is a tool box for modeling, simulation and verification of real-time systems, based on constraint-solving and on-the-fly techniques, developed jointly by Uppsala University and Aalborg University. It is appropriate for systems that can be modelled as a collection of non-deterministic processes with finite control structure and real-valued clocks, communicating through channels and shared variables. A general interface of the program is presented in Figure 3.1

The structure of this program might be divided in three main sections: description language, simulator and model checker. The description language is a non-deterministic guarded command with data types which helps the user in the phases of modelling and design to describe the behaviour of the system as a network of timed automata. The section of simulation and verification are programmed in order to guarantee an automated analysis of the system by the manipulation and the solution of the constraints of the state-space. The simulator enables the analysis of the dynamics of the system during the early modeling stages and provides a mean of fault detection.

- **The modeling framework.** In order to model the system UPPAAL gives to the user both graphical and textual formats for the language used to describe the system. The user has the opportunity to design the system with the Autograph-based graphical interface which might result an easy and immediate way to get used to the program or to deal with simple schemes. When dealing with complex systems described by a large number of element could be useful to work with the textual format. The textual language provides a basic programming language for timed automata.

- **The model-checker.** The model-checker is designed to check invariant and reachability properties, in particular if certain combinations of states and constraints on clocks and integer variables are reachable from an initial configuration. Other properties such as bounded liveness properties can be checked by reasoning about the system in

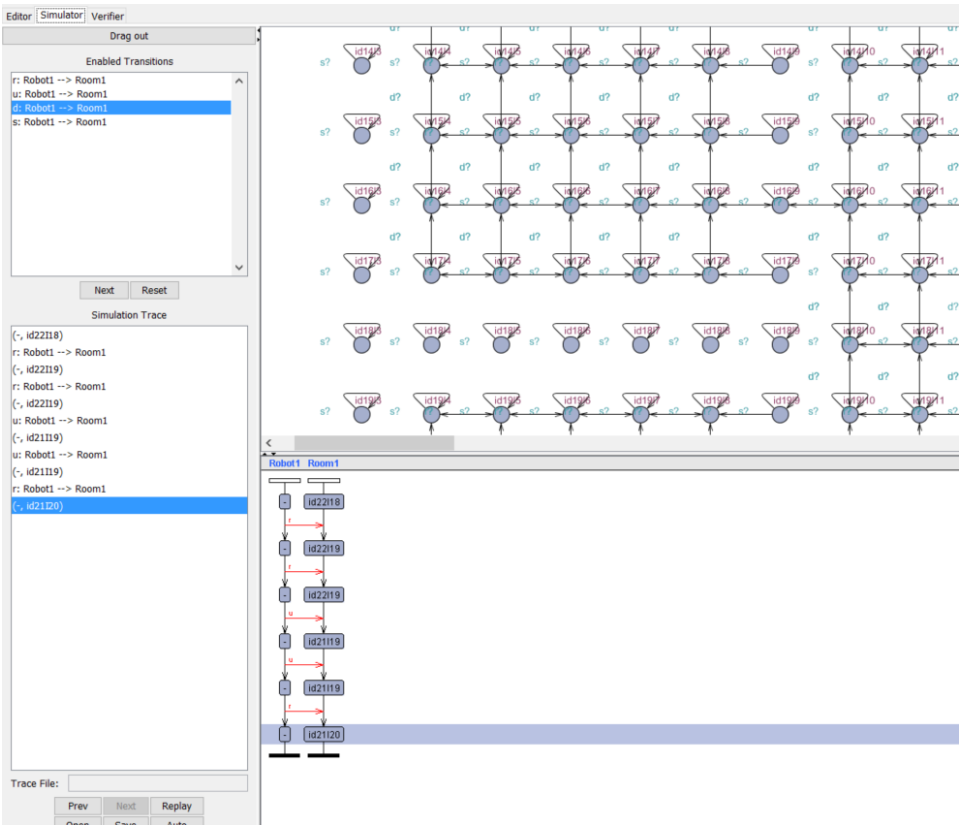

Figure 3.1: Interface of the software Uppaal Model Checker.

the context of testing automata or simply decorating the system description with debugging information and then checking reachability properties.

- **The simulator.** The simulator allows the user to examine in an interactive and graphical fashion the dynamic behaviour of a system. In contrast to the model-checker which explores the whole reachable state-space of a system - examining all the behaviour of the system, the simulator explores only a particular execution trace i.e., a sequence of states of the system. This will in early stages of modelling (or design) provide an inexpensive mean of fault detection. In comparison the model-checker is obviously more expensive as it amounts to an exhaustive simulation covering all behaviour of the system. Another useful application of the simulator is to visualize a diagnostic trace generated by the model-checker; thus the user can in an interactive and graphical fashion examine the execution trace that may result in a system error.

3.3 Experimentation

This section presents a set of tools and robots used in the evaluation of the proposed approach. Section 3.3.1 presents the Nao Robot one of the robots used within our evaluation. Section 3.3.2 a simulation tools that allows evaluating the behavior of the Nao Robot. Section 3.3.3 and 3.3.4 presents the TurtleBot and the Tiago Robot, two additional robots considered during the evaluation.

3.3.1 Nao Robot

Nao [3] is an autonomous, programmable humanoid robot (Figure 3.2) developed by Aldebaran Robotics, a French robotics company headquartered in Paris, which was acquired by SoftBank Group in 2015 and rebranded as SoftBank Robotics. The robot's development began with the launch of Project Nao in 2004. On 15 August 2007, Nao replaced Sony's robot dog Aibo as the robot used in the RoboCup Standard Platform League (SPL), an international robot soccer competition. The Nao was used in RoboCup 2008 and 2009, and the NaoV3R was chosen as the platform for the SPL at RoboCup 2010.

The Nao Academics Edition was developed for universities and laboratories for research and education purposes. It was released to institutions in 2008, and was made publicly available by 2011. Various upgrades to the Nao platform have since been released, including the 2011 Nao Next Gen and the 2014 Nao Evolution.

The various versions of the Nao robotics platform feature either 14, 21 or 25 degrees of freedom. All Nao Academics versions feature an inertial measurement unit with accelerometer, gyrometer and four ultrasonic sensors that provide Nao with stability and positioning within space. The legged versions included eight force-sensing resistors and two bumpers. The most recent version of the robot, the 2014 Nao Evolution, features stronger metallic joints, improved grip and an enhanced sound source location system that utilises four directional microphones. The OS powers the robot's multimedia system, which includes four microphones (for voice recognition and sound localization), two speakers (for multilingual text-to-speech synthesis) and two HD cameras (for computer vision, including facial and shape recognition).

The main specifications are summarised in the following list:

- Hight: 58 cm
- Weight: 4.3 kilograms

Figure 3.2: Nao Robot.

- Autonomy: 90 minutes
- Degrees of Freedom: 25
- Compatible with: Windows, Mac OS, Linux
- Programming Languages: C++, Python, Java, MATLAB, Urbi, C, .Net
- Sensors: Two HD cameras, four microphones, sonar range finder, two infra-red emitters and receivers, inertial board, nine tactile sensors, eight pressure sensors
- Connectivity: Ethernet, Wi-fi

3.3.2 Choregraph

Choregraph is a multi-platform desktop applications, allowing you to:

- Create animations, behaviours and dialogues.
- Perform tests on a simulated robot or directly control a real one.
- Monitor, program and control your robot
- Enrich Choregraph behaviours with your own Python code.

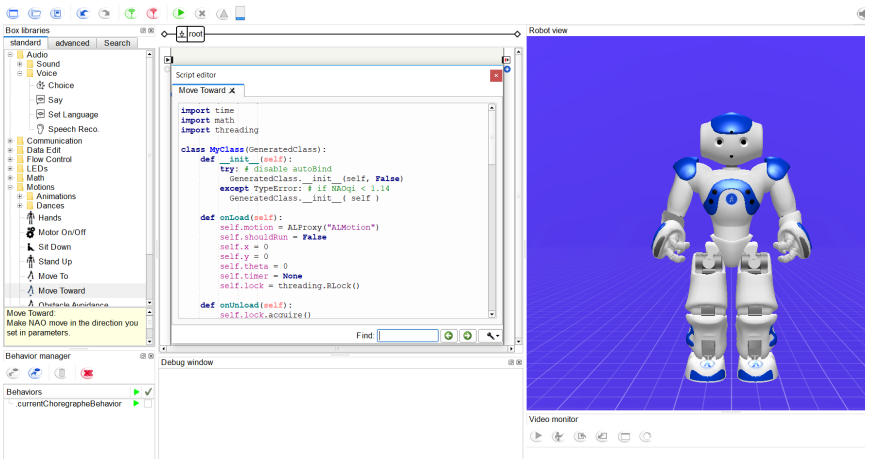

Figure 3.3: Interface of the software simulator Choregraph.

This program enables the user to perform simulations of the expected and programmed robot, it shows in details all the movements executed by the robot in a fictitious environment which can be modelled as the real environment under consideration thanks to a 3D graphical tool. An example of the interface of Choregraph is showed in 3.3

3.3.3 TurtleBot

Figure 3.4: TurtleBot.

TurtleBot [4], presented in Figure 3.4, is a personal robot kit with open source software. TurtleBot was created and developed at Willow Garage by Melonee Wise and Yully Foote in November 2010. With TurtleBot, the user is able to build a robot capable of driving in the environment, see in 3D and have enough power and skills to preform different actions and create various types of applications.

The TurtleBot kit is composed by a mobile base, 2D/3D distance sensor, laptop computer and the TurtleBot mounting hardware kit. In addition to the TurtleBot kit, users can download the TurtleBot SDK from the ROS wiki. TurtleBot is designed to be easy to buy, build, and assemble, using off the shelf consumer products and parts that easily can be created from standard materials. As an entry level mobile robotics platform, TurtleBot has many of the same capabilities of the company's larger robotics platforms, like PR2.

3.3.4 Tiago Robot

Figure 3.5: Tiago Robot.

TIAGO [5], presented in Fugure 3.5 is a moving platform developed by Pal Robotics company. It is a service robot designed to work in indoor environments. TIAGO's features make it the ideal machine for research, especially on ambient assisted or light industry. It combines mobility, perception, ma-

nipulation and human-robot interaction capabilities for one specific goal: to be able to assist in research.

The robot is able to map indoor environments, recognise people and obstacles and perform tasks such as movements and objects grasping thanks to its seven degrees of freedom robotic arm.

Chapter 4

Contribution

This chapter presents the contribution of the thesis. Section 4.1 presents the MEMO framework. Section 4.2 describes how maps containing the building layouts are used to generate Timed Automata (TA) that specify how robots can move. Section 4.3 presents how robots are modeled within MEMO. Section 4.4 describes how missions containing explicit time concerns are specified. Section 4.5 describes how the tool UPPAAL is used to generate plans for the robots. Section 4.6 specifies how MEMO can be used with multiple robots.

4.1 MEMO

MEMO (Map basEd planner for TCTL MissiOns) is a high level planner for robotic applications. An overview of MEMO is provided in Figure 4.1. MEMO takes as inputs an image that describes the environment in which the robots will be deployed, such as the layout of a building, and the models that describe the behaviours of the robots, which may be provided by robotic companies. These elements are used, together with the mission specified by the user, to compute a set of plans (provided as outputs) that ensures the mission achievement and that will be executed by the robots.

Tool Inputs. MEMO takes the following inputs:

- **Image ①.** It contains the layout of the environment. The layout contains a map of the building of interest which represent he floor where the robots will move and operate. Examples of these maps are fire control plans or design documents of buildings. These maps are broadly available in real contexts.

- **Models of the robots ②.** They contain the description of the

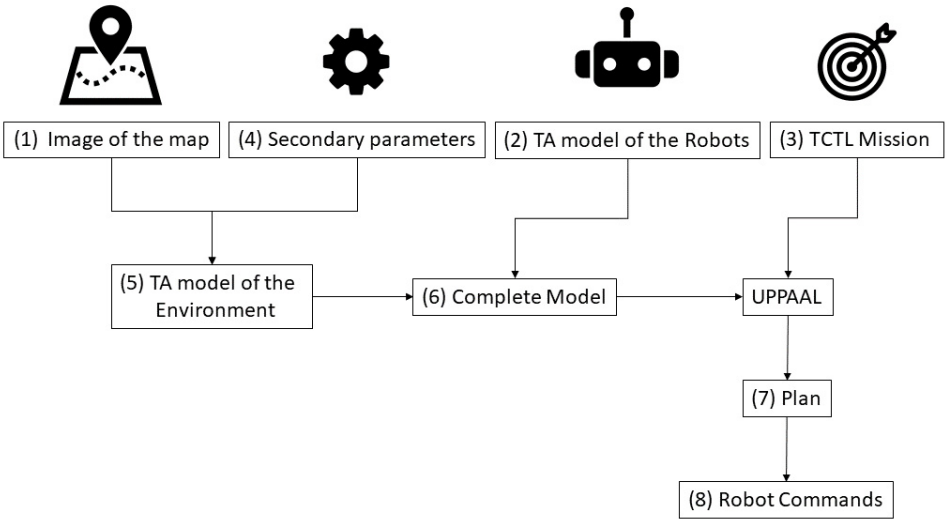

Figure 4.1: A high level overview of the MEMO framework.

possible behaviours of the robot. These models encode freedom degrees of the robots, possible movements and actions. The fact that robot models are provided as inputs to MEMO makes the tool suitable for a large number of different situations where different robot models are considered. The user is enabled to configure the properties and behaviour of his own robot, to program the actions and the sequence of tasks to perform in order to achieve the determined mission. The user has to change parameters such as the diameter of the footprint and the displacement velocity of the models of the robots based on the physical characteristics of the robots that will be deployed in the robotic application. These two values are essential for the creation of the complete model of the system. The first one prevents undesired collision with the walls and any kind of obstacles. The second one, combined with the displacement covered, will keep track of the time necessary for the tasks.

- **Mission of interest ③.** The user, analysing the map or the image of the environment, depending on the the desired mission, should select the initial location of the robots. The user has also to select some "Points of Interest". These points represent the locations where the robots are programmed to perform actions, points that the robot should visit or simply the final configuration at the end of the mission. These information are given to the tool as coordinates (x, y) expressed in pixels, which are easily derived from the image. The mission

the robots must perform is provided by the user as a TCTL formula. Thanks to this logic language the user is able to define specifications that perfectly describes the robot actions and sequences of task and in addition to this he is enabled to specify detailed time constraints using explicit time intervals such as seconds, minutes and hours.

- **Additional parameters ④.** In addition to the main inputs previously described, the user is supposed to provide some other secondary parameters to the tool. In order to create a correlation between the real distances in meters of the environment and the measures of the map, is necessary to set the value of the scale parameter. Another important input to be given to the tool for the creation of the models is the distance with which we will sample and discretize the map. The user insert this parameter in meters and it represents the distance between two consecutive positions in the real building. Since MEMO uses this parameter to discretize the map, it should be selected very carefully. As this parameter decreases the level of details implemented in the map increases; at the same time it reduces the performances in the generation of the discretized map. Another aspect to take in consideration when selection this input is the type of building where the robot is moving. If the user wants to perform simulations and verification on a large warehouse with few obstacles a large sampling factor can be chosen. Vice versa, if the building of interest has tens of rooms, offices, doors and corridors, a small value of the sampling factor allows the model to describe important details that are necessary for a correct planning. How to find a good value for the sampling factor, and providing automated procedure for its selection, is out of the scope of this thesis.

MEMO in Action. MEMO uses the inputs previously described to compute plans that ensure the satisfaction of the desired mission. These plans are then performed by the running robots. Specifically, MEMO works as follows:

- **Encoding the image into an equivalent TA ⑤.** Our tool takes the image of the map of the environment and, thanks to the developed algorithms, converts it in the equivalent Timed Automaton model of the system environment. While performing this operation the algorithm evaluates the secondary parameters given as inputs by the user.
- **Creation of the model of the system ⑥.** The Timed Automaton describing the environment and the models of the robots are combined

to obtain a model of the system. Thus, the obtained model of the system contains both the model of the environment and the models of the running robots.

- **Plan creation ⑦.** The model of the system together with the mission specified by the user is given as input to UPPAAL Model Checker. With this program is possible to perform both simulation and verification. The user is able to select the possible actions to the robot e simulate its behaviour and its movements. UPPAAL analyses them and automatically verifies if the robot is able to achieve the mission in the given time interval. If the specified mission is feasible according to the environment and to the time constraint, the model checker gives as output a trace representing the sequence of the states and transitions followed by the robot from the initial to the final configuration in order to fulfil the task.

- **Deployment of robot commands from the plan ⑧.** The trace produced in step ⑦ contains the trace to be followed by the robots to accomplish the mission. The trace contains the sequence of states that define the points on the real map with respect to the global frame of the building. MEMO analyzes the trace, parses it and compute the actions that the robots should perform. The computed actions are forwarded to a simulator or to the running robots.

4.2 Encoding the Environment

One of the main advantages of MEMO compared with planner proposed in state of the art, is that it is able to compute plans from real maps that simply contains images of the building where the robots will be deployed. These maps may contain layout of building, such as offices, warehouses or hospitals. Maps are used to create a model that takes into account all the physical obstacles as walls, corridors and doors. An example of map layout representing the Jupiter Building of the University of Gothenburg is presented in Figure 4.2.

4.2.1 Preliminaries

MEMO creates a model from a continuous space by discretizing the image in a predetermined amount of points from which it is possible to describe the fundamental characteristics of the map.

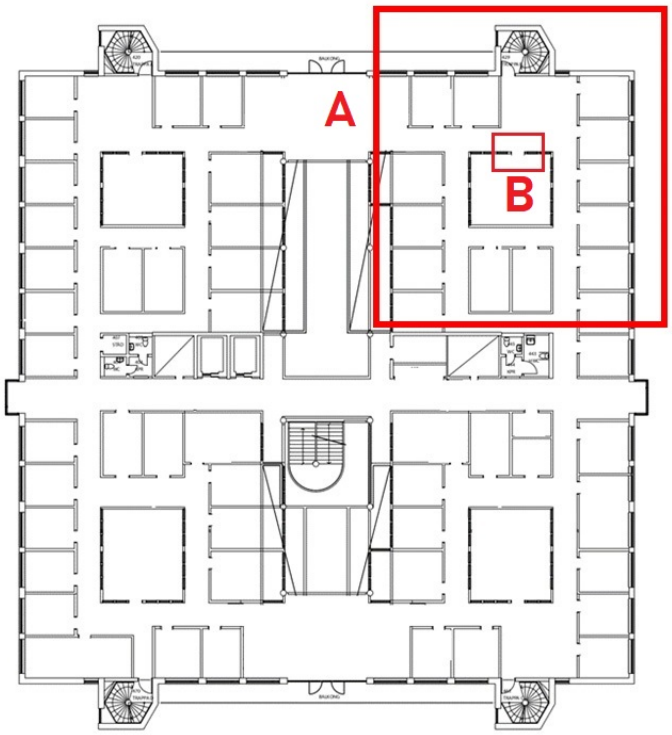

Figure 4.2: An example of map layout. It represents the map of the Jupiter Building of the University of Gothenburg.

The span parameter. The distance between points is defined through the *span* parameter. The *span* parameter is provided by the user and defines the level of details with which we create the model. The lower is the value of this parameter, the higher is the resolution of the sampling. Choosing the right value for the *span* parameter is not always easy, since on one hand we would like to have a fast generation of the TA and thus to have a high *span* value, on the other hand we would like to have precise results, and thus keep the value of the *span* parameter low. How to select a good value for the *span* parameter is out of the scope of this thesis and will be addressed in future work. However, to ensure correctness, the span must ensure the property 4.1, i.e., the span must be smaller then the difference between the door width and the diameter of the robot divided by two. This relation ensures that doors of the plan are not neglected and thus rooms will not result to be unreachable.

$$span < (doorwidth - robotdiameter)/2 \tag{4.1}$$

Discretizing the map. MEMO discretizes the map into a large rect-

angular matrix that preserves the proportion of the image. The measures of the width (W) and the heigh (H) of the environment and the *span* parameter (expressed in meters), define the size of the matrix to be generated. Considering the width (W) of the environment on the X axis and the heigh (H) along the Y axis, to represent the positions on the map we define the sets X and Y as specified in equations 4.2 and 4.3

$$X = \{x \in \mathbb{N} \mid 0 \leq x \leq W - span, (x \ \% \ span) = 0\} \tag{4.2}$$

$$Y = \{y \in \mathbb{N} \mid 0 \leq y \leq H - span, (y \ \% \ span) = 0\} \tag{4.3}$$

The sets X and Y contain all the possible X and Y coordinates within the considered map. Then, every discrete position in the map is identified by a point (x, y) where $x \in X$ and $y \in Y$. The values of these two sets will be fundamental for the algorithms when defining the constraints representing the walls and obstacles in the environment. Indeed during all the procedure the algorithm always keeps a direct association between the discretized point just computed and the relative points on the map on the figure.

Let us consider the set of actions $Act = \{u, d, l, r, s\}$, representing possible movements an agent (i.e., the robot) can perform in the environment, where actions u, d, l, r represent the fact that an agent can move up, down, left and right, respectively. For every $a \in Act$, let $Block(a)$ be the set of positions (x, y), with $x \in X$, $y \in Y$, where the robot cannot take action a when its current position is (x, y) due to the detection of an obstacle or wall between the source and target point. For instance, $Block(u)$ are those positions (x, y) from which the robot cannot move to reach $(x, y - span)$.

The two encodings. MEMO consider two encoding to represent the considered map as a Timed Automaton. The general idea behind the two algorithms is similar. The algorithms checks whether it is possible to reach one point from another by checking for the presence of obstacles, i.e., walls and doors. If it is possible to go from one point to another a transition is added between the two points. From a usage perspective, the models generated from the two encodings are equivalent. The two encodings are presented in the following.

4.2.2 Encoding 1

MEMO encodes the points of the environment through a set of Timed Automaton states. The Timed Automaton encoding the environment is defined as follows:

- Each state s of the Timed Automaton represents a point (x, y) of the map.

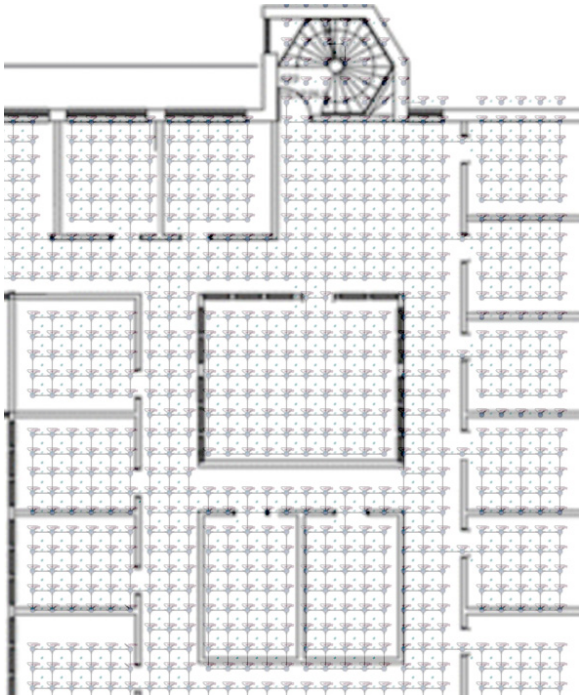

Figure 4.3: Timed Automata model of the considered region of the Jupiter building.

- Consider two state s_1 and s_2 representing two points (x_1, y_1), (x_2, y_2) such that $|x_1 - x_2| = span$ and $|y_1 - y_2| = 0$ or $|x_1 - x_2| = 0$ and $|y_1 - y_2| = span$, transitions encodes whether it is possible to move from (x_1, y_1) to (x_2, y_2). A transition from s_1 to s_2 is in the timed automaton if it there is no obstacle (e.g., no wall) between state s_1 and state s_2. A self-loop transition is also added on each state of the environment, meaning that the robot can remain its current location.

- To enable synchronization between the model of the robots and the environment in which they are deployed transitions are labelled. Specifically, that depending on the direction would be labelled as "u?" (up), "d?" (down), "l?" (left) and "r?" right. The self-loop on each state is labelled with the character "s?" (stay).

For example, the TA generated from the portion of the layout of the Jupiter building indicated in Figure 4.2 with the symbol A, is presented in Figure 4.3. As evidenced in the figure, states represent locations of the environment, transitions specify how it is possible to move from one state to another.

In the following we formally describe how the Timed Automaton (TA) that describes the environment is defined. Specifically, a Timed Automaton

(TA) $\mathcal{E} = \langle Q, q_{x_0,y_0}, v^0, I, L, T \rangle$ that describes the environment is defined as follows:

- $Q = \{(x, y) \mid x \in X, y \in Y\}$ represent the possible positions of the robot on the map;
- $(x_0, y_0) = q_{x_0,y_0}$, such that $q_{x_0,y_0} \in Q$ is the initial position of the robot;
- T is a set of transitions such that each transition $t \in T$ satisfies one of the following rules:
  - $(q_{(x,y)}, q_{(x,y-span)}, \emptyset, \emptyset, (u, ?), \emptyset, \emptyset) \in T$ if and only if $(x, y) \notin Block(u)$;
  - $(q_{(x,y)}, q_{(x,y+span)}, \emptyset, \emptyset, (d, ?), \emptyset, \emptyset) \in T$ if and only if $(x, y) \notin Block(d)$;
  - $(q_{(x,y)}, q_{(x-span,y)}, \emptyset, \emptyset, (l, ?), \emptyset, \emptyset) \in T$ if and only if $(x, y) \notin Block(l)$;
  - $(q_{(x,y)}, q_{(x+span,y)}, \emptyset, \emptyset, (r, ?), \emptyset, \emptyset) \in T$ if and only if $(x, y) \notin Block(r)$;
  - $(q_{(x,y)}, q_{(x,y)}, \emptyset, \emptyset, (s, ?), \emptyset, \emptyset) \in T$ for every position x, y.

Intuitively, a robot can go up, down, left and right if and only if the current state (x, y) is not in the set $Block(u)$, $Block(d)$, $Block(l)$, and $Block(r)$, respectively, meaning that the path that moves to position $(x, y-span)$,$(x, y+span)$, $(x - span, y)$ and $(x + span, y)$ is not blocked.

For example, the TA generated from the portion of the layout of the Jupiter building indicated in Figure 4.2 with the letter B, is presented in Figure 4.4. As evidenced in the figure, it is possible to reach the set of states framed by a box marked with symbol ① from the states framed by a box marked with symbol ② only by crossing states that encode points located in correspondence with a door.

4.2.3 Encoding 2

MEMO encodes the points of the environment through a single Timed Automaton state and a set of transition. The main idea behind the second encoding of the environment is the following.

- The points of the environment are modelled through two integer variables x_{rob} and y_{rob}, with $0 \leq x_{\text{rob}} \leq w - span$ and $0 \leq y_{\text{rob}} \leq h - span$.
- Transitions specify how agents can move within the environment by changing the values of variables x_{rob} and y_{rob}.

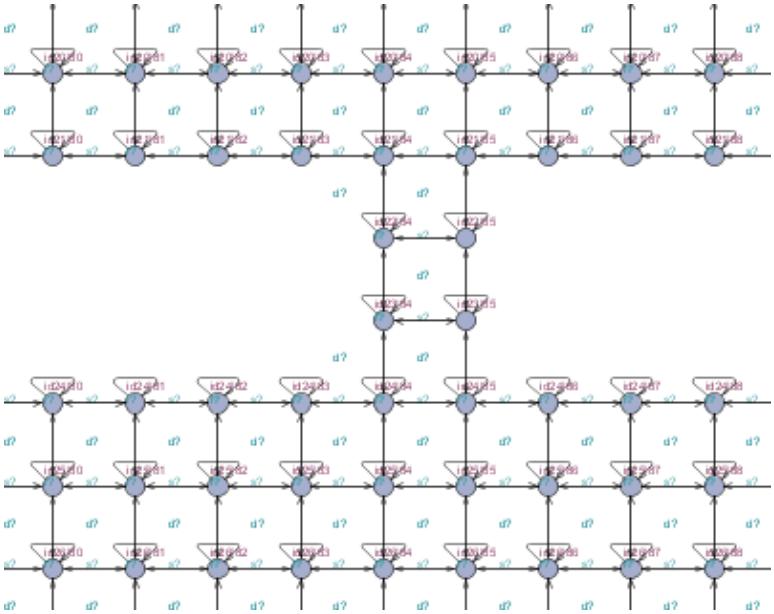

Figure 4.4: Detail of a door represented in the Timed automata model.

The fact that a single state is used to represent the environment makes the encoding more compact with respect to the number of states.

In the following we formally describe how the Timed Automaton (TA) that describes the environment is defined. For every $a \in Act$, let γ_a be the formula $\bigwedge_{(x,y)\in Block(a)} \neg(x = x_{\text{rob}} \wedge y = y_{\text{rob}})$. Hence, γ_a is true when the current position of the robot is not a position in s. Then, a Timed Automaton (TA) $\mathcal{E} = \langle Q, q_{x_0,y_0}, v^0, I, L, T\rangle$ that describes how an agent can move within its environment is defined as follows:

- $Q = \{q\}$ contains a single state that encodes all the locations of the environment;
- $q_0 = q$ the only state of the environment is also the initial state of the automaton;
- T is a set of transitions such that each transition $t \in T$ satisfies one of the following rules: Transitions are defined as follows, for any $(x, y) \in X \times Y$:
  - $(q, q, \gamma_u, \emptyset, (u, ?), \emptyset, y_{\text{rob}} = y_{\text{rob}} - span) \in T$
  - $(q, q, \gamma_d, \emptyset, (d, ?), \emptyset, y_{\text{rob}} = y_{\text{rob}} + span) \in T$
  - $(q, q, \gamma_l, \emptyset, (l, ?), \emptyset, x_{\text{rob}} = x_{\text{rob}} - span) \in T$

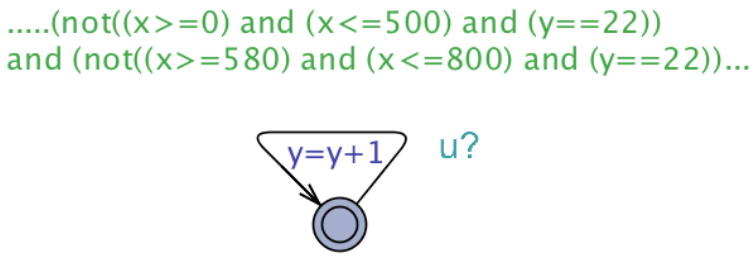

Figure 4.5: Portion of the model of the environment represented with Encoding 2.

- $(q, q, \gamma_r, \emptyset, (r, ?), \emptyset, x_{\text{rob}} = x_{\text{rob}} + span) \in T$
- $(q, q, \emptyset, \emptyset, (s, ?), \emptyset, \emptyset) \in T$

The proposed encoding has a single state and one transition for each direction of the motion. Intuitively, the possible movements of an agent within its environment are encoded through five transitions that specify when the agent can go up, down, left and right and the transition stay. Every transition is labelled with

- a *guard* specifying the condition that enables the robot to move in a given direction. More precisely, the guard list all the coordinates for which is possible to execute an action. For example, the transition labelled with u contains all the points of the environment that are not in the set $Block(u)$.
- the *update* of the coordinate that is going to change because of the move. When going up, down, left and right the current position of the agent is updated as follows $y_{\text{rob}} = y_{\text{rob}} - span$, $y_{\text{rob}} = y_{\text{rob}} + span$, $x_{\text{rob}} = x_{\text{rob}} - span$ and $x_{\text{rob}} = x_{\text{rob}} + span$). When performing the action stay is performed, the variables x_{rob} and y_{rob} encoding current position of the agent are not updated.

For example, in Figure 4.5 is represented a portion of the TA obtained from the layout of the Jupiter Building.

As evidenced in figure, since the states with $x \geq 0$ and $x < 500$ and $y = 22$, and $x \geq 580$ and $x < 800$ and $y = 22$ are not in $Block(u)$, a self-loop transition is present that increments the value of the variable y.

Comparison with encoding 1. Encoding 2 is equivalent with respect to Encoding 1 in terms of usage. In encoding 2, variables x_{rob} and y_{rob} are used to all the states of the TA of encoding 1. Even if encoding 2 has less state and transitions, the guards of the transitions are labelled with large

arithmetical formulae which must be evaluate to check when transitions between states can be fired. In contrast, encoding 1 has more states and transitions. However, transitions are not labelled with guards, that is guards are handled by changing the current automaton state. However, in terms of usage, the logic that triggers synchronization among labels "u?", "d?", "l?", "r?" is exactly the same, and so the two encodings are interchangeable.

4.3 Encoding the robot

MEMO assumes that the robot is modelled through a TA. In an ideal scenario, the TA can be provided by robots vendors, or template models can be downloaded as off-the-shelf models and then customized by the final user. Independently on how this model is created, the TA model of the robot must model two aspects of the robots that are of interest for the plan computation:

- *robot movements concerns.* It tries to describe how a robot can move in its environment. It describes how a robot can move in the different directions based on its kinematics aspects. For example, it contains the maximum velocity of the robot.

- *task execution.* It refers to the ability of the robot in performing tasks in different points of the environment. Tasks represent actions a robot can perform, such as "clean the floor". With respect to action execution, the model should also describe the time required for the execution of every action.

Encoding robot movements concerns. Two aspects of movement are encoded within the model of the robot: how a robot can move and time related aspects of its movement.

- *Movement.* Depending on the type of the robot, different models of robot movement can be provided. Consider for example the model presented in Figure 4.6. Movement concerns are represented through five locations labeled with s, u, d, l and r representing the different movement directions of a robot. When the robot decides to move in one direction the current location changes and one of the four states is entered depending on the direction the robot wants to follow. Every transition is labelled with an event that is used to synchronize the model of the robot and the one of its environment. If the robot decides to go up, it has to fire a transition labeled with the event "u!". This transition can be fired only if a transition labeled with the event "u?"

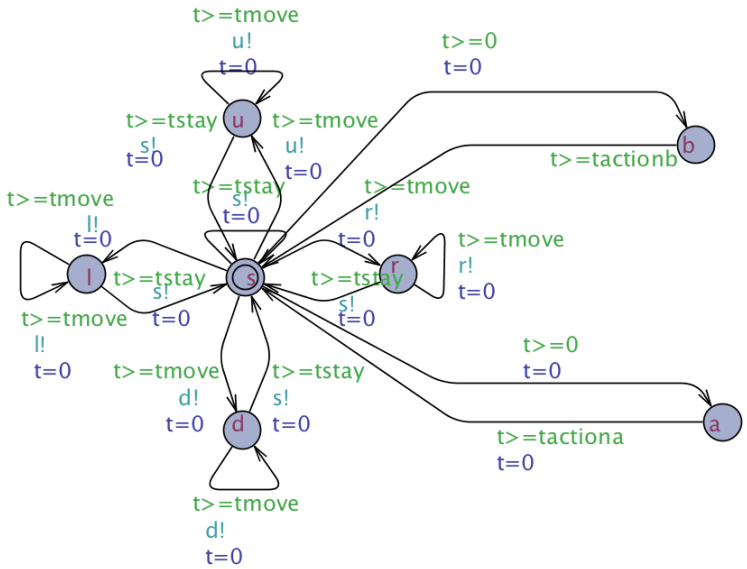

Figure 4.6: An example of TA robot model able to move in four different directions and perform two different actions.

is enabled in the environment. This procedure synchronizes the model of the robot and its environment.

- *Time related aspects.* We describe how time related concerns are encoded in the model of the robot. The TA describing the behaviour of the robot is defined over the set of clocks $C = \{t\}$, which is used to model how the time passes when robot perform movements and action. Let *speed* be the maximum speed of the robot and recall that *span* is the value used to create the discrete map of the environment. Then, the time *tmove* needed to perform a movement is computed as specified in equation 4.4. In other words, *tmove* is the minimum time required to cover a distance of length *span*.

$$tmove = \frac{span}{speed} \tag{4.4}$$

The value *tmove* is used in the labelling of the transitions to encode how time passes when actions are performed. Specifically, the value *tmove* is used in the guards $t \geq tmove$ to set the duration of each transition labelled with action u, d, l and r, meaning that it takes to the robot at least *tmove* to move up, down, left and right, respectively. When a transition is performed, in order to manage the time delay of every action during all the motion, the clock t is reset to zero and the

robot is forced to wait in the current configuration until when, with the increase of the clock value another transition is enabled again.

The value *tstay* represents the time the robot needs to stop, and must be set by the user depending on the characteristics of the robot. For this reason, the transitions leading to the location "stay" are labelled with the guard $t \geq tstay$.

The synchronization among the environment and the robot ensures that if the robot wanted to perform an action that would lead to collide an obstacle (e.g., the wall), it can not perform such transition since the environment does not sync with the robot. Consider for example the case in which a robot wants to go up, i.e., it wants to perform a transition labelled with $u!$. A transition of the environment labelled with $u?$ must be synchronously fired. However, to have a transition of the environment labelled with $u?$ enabled, an obstacle should not be present on the trajectory of the robot.

Encoding robot actions. The model of the robots may also describe arbitrary actions the robot can perform. Actions may encode different activities, such as "load-unload an object", "change or charge the batteries", "open a door". We now present how actions are encoded in the model of the robot.

- *Encoding actions.* To execute an action, a robot must move in an appropriate state in which the action is actually performed. For example, state a in Figure 4.6 is a state in which an action is performed. We assume that before performing an action, a robot must be stopped. For this reason, state a is only connected to state s.

- *Time related aspects.* Time related aspects associated with the execution of an action are modelled by acting on the guards of the transitions. Specifically, consider the state a in Figure 4.6 in which the robot is performing the action. The Timed Automata model is forced to remain in state a, until the condition $t \geq t_{actionO}$ holds, i.e., an amount of time greater than or equal to $t_{actionO}$ should pass before the robot goes back to the idle state.

4.4 Encoding the missions

Missions specify what the final goal a robot application should achieve. Since, we assume the usage of UPPAAL as a software component in charge of computing plans, missions are specifies as reachability properties. This kind of property concerns:

- The execution of actions. For example, the user could request that an action must be finally performed by the robot.

- The existence of a path on the map that from the initial configuration leads to specific locations programmed by the user. For example, a robot should finally reach a location, possibly within a given time limit.

To be able to specify properties that contains explicit time concerns, we define a *global clock* c in the Timed Automata network that contains the model of the robot and its environment. Unlike the clock that manages the time constraints on the transitions that is reset to zero every time an action is executed, this clock is never reset. The task of the global clock is to keep track of the global time during the entire development of the mission and it is necessary to have a reference for the time constraints expressed in the mission the team of robots must achieve.

Handling actions. Since the mission of interest predicates on the occurrence of specific actions or events that encode that a robot reached a location, the model of the robots and its environment are slightly change based on the considered mission. In order to register the fact that an action has been performed we define a boolean variable flag_a, showed in Figure 4.7, initially set to zero for any action $a \in Act$.

Our algorithm is able to identify during the development of the mission when an agent is located in one of the Points of Interest and it is ready to perform the correspondent action, in this configuration in the model of the robot the transition leading to the action location are enabled and after waiting the necessary amount of time, the relative boolean variable flag_a is set to one.

The previous step is very important in for defining the robotic mission. After performing the described changes in the Timed automata, in order to request the achievement of a determined action is enough to ask that flag_a to be $True$.

Handling location reachability. To deal with reachability of points on the map, the model of the environment presented in previous section must be modified according to the inputs of the user before the creation of the environment of the model. In the first encoding, all the transitions that enter the state that represents location (x, y) set variable $l_{(x,y)}$ to true. In the second encoding, when the variables x_{rob} and y_{rob} are set to values (x, y), the variable $l_{(x,y)}$ is set to true.

- *Performing an action or reaching a location in a time bound.* Introducing now the time bounds in the specifications and combining them

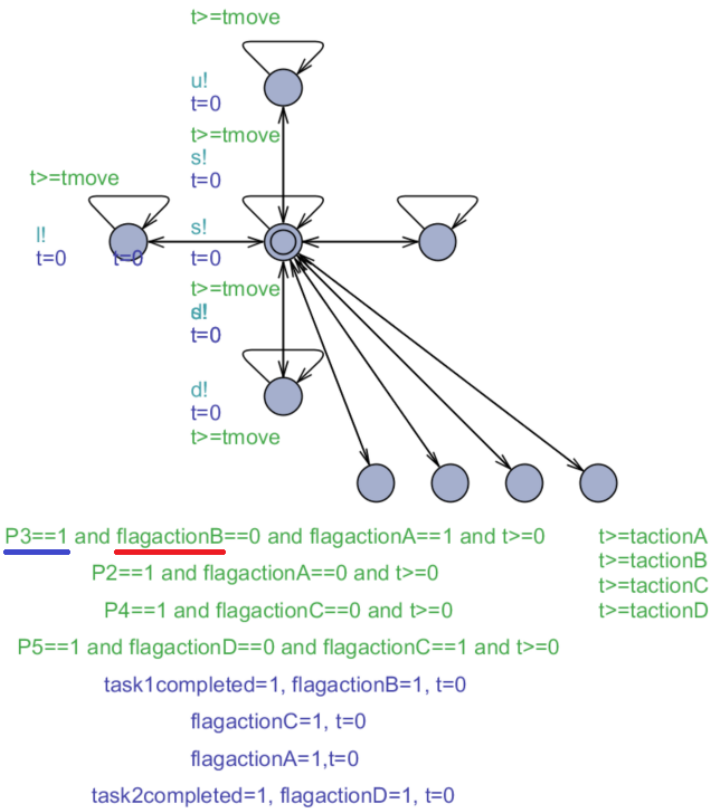

Figure 4.7: Boolean variables implemented on the transition of the robot in order to specify the behaviour of the robot.

with the boolean variable introduced above, we now describe how to express that an action a must be achieved within a time bound t_{bound}. The specification is expressed as shown in 4.5:

$$E\Diamond(flag_a \wedge c < t_{bound})equation \tag{4.5}$$

- *Reaching a location within a time bound.* Another possible specification is the one that verifies if the robot is able to reach a determined location (x, y) within a determined interval of time t_{bound}. Let $l_{(x,y)}$ be a boolean variable that the algorithm sets to one when the robot enters the location (x, y). The formula 4.6

$$E\Diamond(l_{(x,y)} \wedge c < t_{bound}) \tag{4.6}$$

expresses that the robot is supposed to visit the location (x, y) earlier than t_{bound} from the beginning of the motion, clock c keeps track of the real time during the simulation.

- *Performing a set of actions or reaching a set of locations in a time bound.* Sometimes the mission may require a robot to perform a set of actions $\{a, b, c\}$ within a specific time bound. Hence, the property can be expressed by the formula 4.7

$$E\Diamond(\bigwedge_i flag_i \wedge c < t_{bound}),\ with\ i \in \{a, b, c\}. \tag{4.7}$$

- *Ordered sequence of actions.* In this section we present how it is possible to define more complex specifications concerning several actions that must be executed by the robot in a well defined order. This feature represents a general case of a robot behaviour which it it easily scalable to several real scenarios in which the robot is programmed to perform a set of ordered actions in order to achieve a complex task. It is possible to implement this characteristic with little modifications in the robot model, where the user is supposed to set the desired sequence of actions and the right order. Considering actions a, b, c that must be performed in this order. In the Timed Automata model this three actions are implemented as three new locations, in order to express the constraints of the order for each state we define a boolean variable $flag_a$, $flag_b$ and $flag_c$ initially set to zero and updated to one after the execution of the correspondent action. In this way it is possible to set logic conditions on the transitions leading to the actions, in which an action cannot be executed before the previous one. The property that expresses this specifications is showed in 4.8:

$$E\Diamond(missionAchieved \text{ and } c < t_{bound}) \tag{4.8}$$

  where *missionAchieved* is another boolean variable set to one after the execution of the last action. Programming in this way the robot model and analysing the specification described above, in order to be verified the simulation is forced to perform all the actions in the right order before the time expires.

4.5 Using UPPAAL to generate and simulate plans

In this section we show how UPPAAL is used by MEMO to compute, analyse and simulate robotic plans. Specifically, we will discuss how UPPAAL can be used to compute plans, and simulate the execution of a given set of plans.

- *Computing plans.* The UPPAAL model checker can be executed by providing as input a network of TA describing the model of the robot

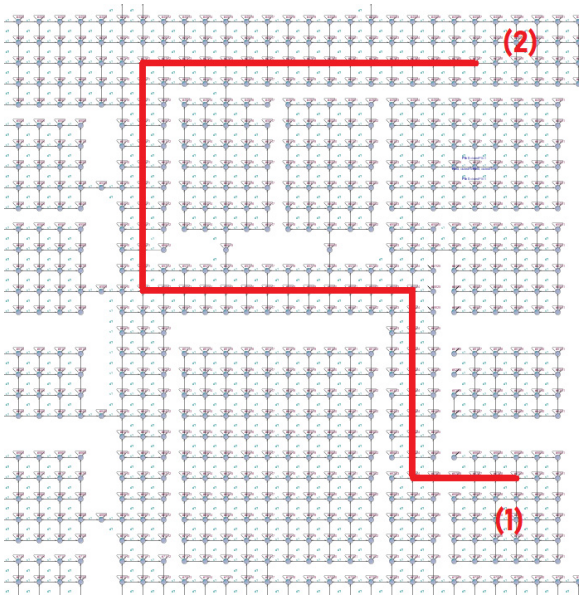

Figure 4.8: Plan computed by MEMO represented on the TA model of the environment.

and its environment, and a mission to be achieved to generate a plan that ensures the satisfaction of the mission of interest. The trace (counterexample) produced by UPPAAL describes a plan that ensure the mission satisfaction. The plan contains the locations and actions that the robots have to perform in order to achieve the desired mission. MEMO exploits UPPAAL to automatically compute the commands that the robots must execute. This approach guarantees fast and automatic results without any need of any user corrections or operations during the process. MEMO automatically delivers the trace created from the input specifications to the robots.

In Figure 4.8 we represent the trace computed by our tool, considering a general mission of reaching the location 2 starting from an office defined as location 1. The trace is composed by the sequence of the states visited along the path connecting the starting and final locations.

- *Simulation.* UPPAAL also allows the user to simulate the behaviour of the robots when a plan is performed. In this case the user after selecting the desired parameters and after the creation of the complete Timed Automata model, is enable to open the system through the graphical interface of UPPAAL and manually simulate the execution of a plan. Following this approach the user can select the next action a robot can perform and manually compute a trace that achieves the

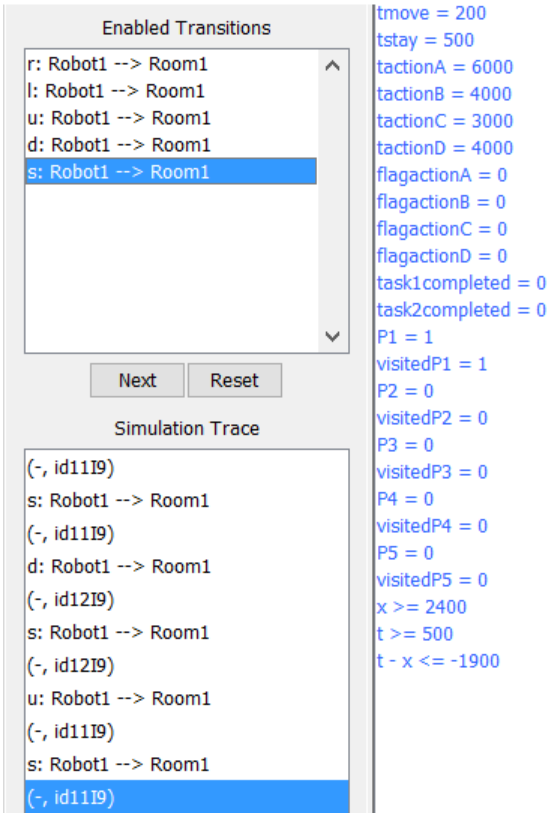

Figure 4.9: Uppaal interface for the simulation of the of the robot behaviour.

satisfaction of mission of interest. Despite a great customizability and freedom of choice, this approach is not automatic.

Figure 4.9 presents a perspective of the UPPAAL tool that enables the user to manually simulate the behaviour of the robot. At each time instant it shows the enabled transitions that the robot can perform in the current configuration. At each execution of a transition the list of enabled transitions is updated concerning the new constraints of the model. While simulating the development of the mission UPPAAL keeps track of all the variables of the system and an execution trace is generated.

In the following we provide some additional remarks related with the automatic computation of plans performed by using the UPPAAL model checker.

- The fact that none of the locations have an invariant and that transitions are labelled with guards that only require that $t \geq tstay$ and

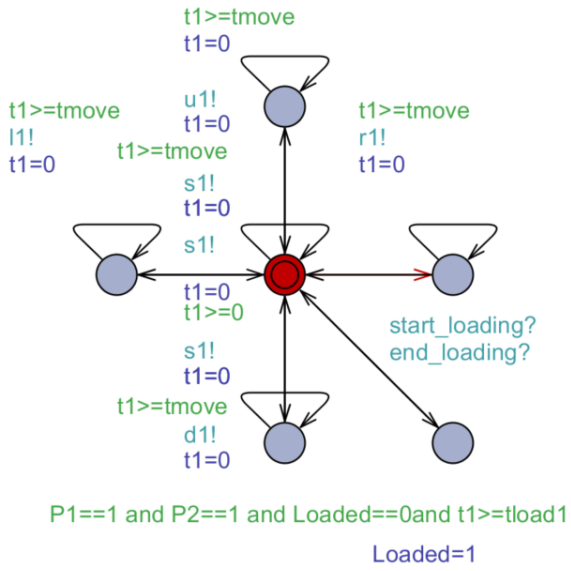

Figure 4.10: Portion of a robot model that belongs to the team.

$t \geq tmove$, allows a robot to move from a position to another one with an infinite amount of time because transitions leading to a location labelled with a motion action. However, the goal of the mission, that is expressed in the planning properties verified through the solver, includes timing constraints that limit the duration of the entire operation.

- A plan can also be not present. The maximum speed of the robot and the time duration of the action execution may make the desired mission is unfasible for the designed robot. In this case, the UPPAAL model checker returns a value that indicates that no plan is available.

- In this work we are not trying to find the best path or the one that requires less time, we are just searching for a plan that ensure the satisfaction of the mission of interest.

4.6 Handling multiple robots

The main characteristic of a team of robots is the ability to collaborate and help each other in order to achieve complex missions and tasks that would be impossible to achieve for a single robot working alone. Our tool proposes a solution to the creation of a plan that includes more than one robot.

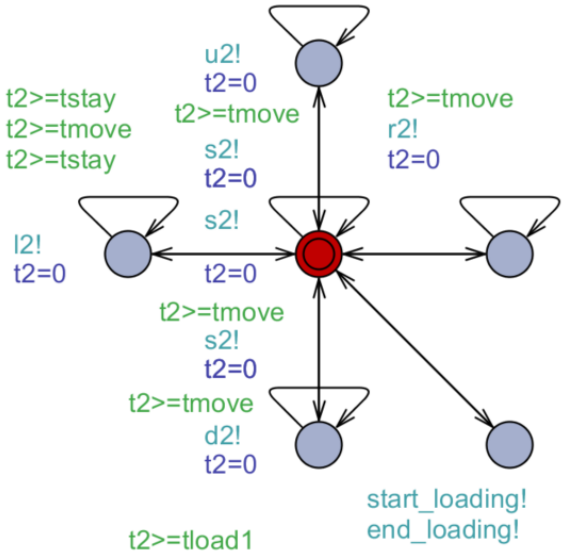

Figure 4.11: Portion of a robot model that belongs to the team.

Encoding multiple robots. The main idea is that each robot of the team is associated to TA modelling the robot and a TA modelling its environment. The models of the robots are strongly interconnected, since they synchronize among each others when collaboration is needed for action execution. The complete model could be seen as the iteration on n single robot models, where n is the number of robots present in the team. As in the single robot scenario the guards on the transitions of the robots constrain the values of local clocks to manage the time needed to perform movements and actions. A global clock, common to all the agents of the team which keeps track of the overall time for the specifications.

For example, Figures 4.10 and 4.11 represent a team of two robots programmed to perform a collaborative action. Collaboration is achieved by synchronously executing the transitions labeled with *start_loading*? and *start_loading*! and *end_loading*? and *end_loading*!.

The models of the robot are very similar to the ones described in the previous case, while the model of the environment, since is unique in the system is completely different. On the single state of the environment for each of the n robots of the team we implement the five actions needed to describe the robot behaviours each one associated to the relative subscript going from 1 to n, "u_n", "d_n", "l_n", "r_n",and"s_n".

Regarding the encoding of the environments the following changes have been applied

- *Encoding 1.* Since we have an environment model for each robot, in order to let them communicate, for each robot we implement a pair of global variables that represent the spacial coordinates of each robot. These variables are used to keep track of the movements of the agents at every time instant. These variables are exploited in the conditions that guarantee the collision avoidance between the robots. In order to do avoid interference between the different models of the environment, the transition of each sub-system are labelled with respect to the relative robot.

- *Encoding 2.* This encoding adopts a completely different strategy with respect to Encoding 1 in order to deal with a team of robots. Since in the single robot model the variables representing the spacial coordinates are already implemented, it is possible to rely on only one model per robot for the environment.

Encoding synchronization among robots. When dealing with two or more robots that are supposed to work on the same environment and collaborate, we must consider the conditions and the commands that address synchronization among robots. Consider for example a condition in which we want a robot to load another robot. To ensure synchronization when actions are executed, in each of the models of the robots location named "load" is created and linked with the "stay" location by means of two transition. In one robot, that could be seen as the "master", we add the action *load*! on the transition that reaches the state load. On the second robot, that could be seen as the "slave", we add the action *load*? on the transition that reaches the state load. This mechanism ensures that the transitions must be synchronously fired among the two automata.

As showed in 4.12 the two robots start from two different locations 1 and 2 and following the instructions given by the plan computed by MEMO, they perform the trajectories a and b, they synchronise in the location 3 and they perform a collaborative action.

Check for collisions. To check for collisions it is necessary that the two robots do not reach the same location of the environment at the same time. Thus, the mechanism to detect collisions strongly depends on the encoding used for the environment. We discuss how to implement a procedure to check collision avoidance for the Encoding 2, but a similar approach can be used for Encoding 1.

As previously discussed, each model of the environment of the robot n has two integer variables $(x_{\text{robn}}, y_{\text{robn}})$ that identify in which state the robot n is located at the current time instant. At this point the main idea of

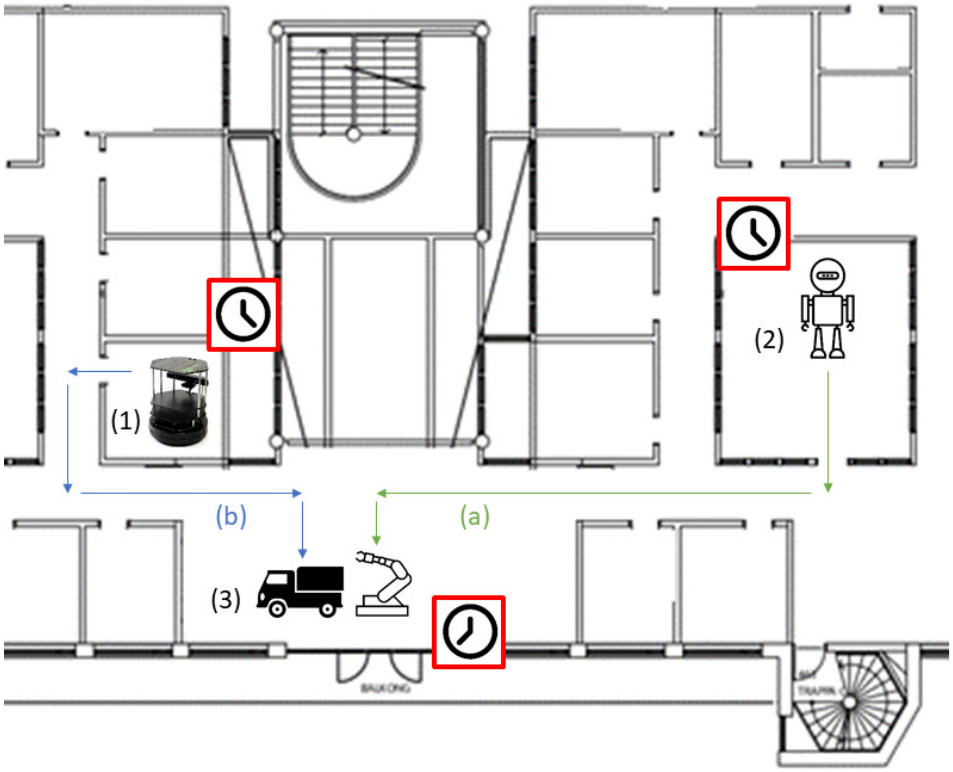

Figure 4.12: A team of two robots performing a collaborative and synchronised action.

the implemented feature of collision avoidance is to limit the movements of the robot, if the agent is going to perform a displacement towards a determined position the algorithm verifies in real time if the target location is occupied by another robot. If the location is not occupied the transition is enabled, otherwise the tool behaves as if there were an obstacle on the path. Note that UPPAAL fires one transition at the time. Execution of multiple transitions during the same time instant is obtained by considering zero time transitions.

Considering a general case in which the robot 1 aims to perform the action "right" avoiding any kind of collision with robot 2. Implemented in the "right" transition of the first agent the constraints to enable the displacement is described by 4.9:

$$(x_{\text{rob1}} + 1 \neq x_{\text{rob2}} \; and \; y_{\text{rob2}} \neq y_{\text{rob2}}) \tag{4.9}$$

This procedure checks if the target location of the first robot, which is obtained incrementing the x coordinates, does not coincide with the current one of the second robot. This method is applied to all the transitions of

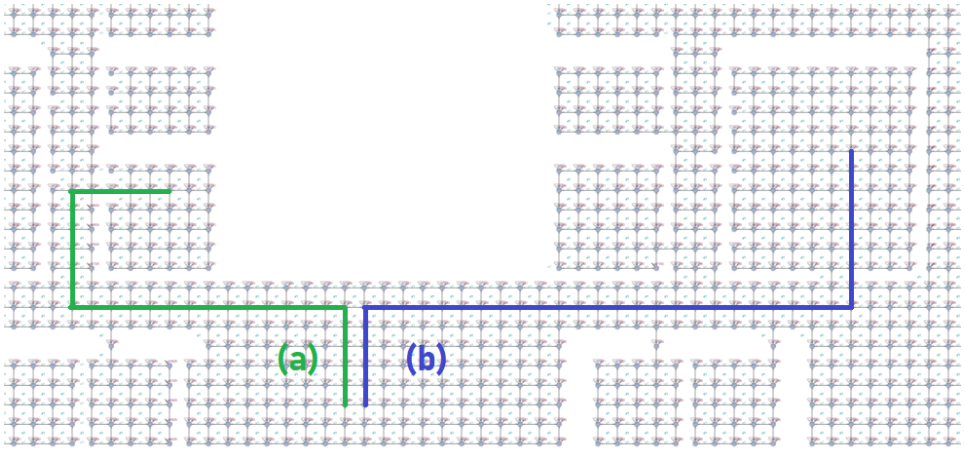

Figure 4.13: Plan computed by MEMO for a team of two robot represented on the TA model of the environment.

the robot except the action "stay", which does not require any limitations. The constraints applied to each robot regards their target location with respect to the others current location, this combinations cover all the cases of multiple robots moving in the same environment. The same procedure can be applied with teams containing more than two robots are considered.

Mission specification. As explained in the previous chapter, the logic behind how a robot organises its movements, sequences of ordered actions and synchronization is managed by the conditions applied to the transitions of the model of the robot. In the case of multiple robots scenario the transitions manage also the synchronisation between the agents. We implemented a boolean variable (e.g. *Loaded*), initially set to zero, that keeps track of the collaborative actions performed. If two robots, after meeting at the same location at the same time, fulfil a common action, this variable is set to one.

Defining a specification such 4.10, in order to fulfil the mission, the system is forced to satisfy all the conditions and configuration that lead to set the variable *Loaded* to one, and therefore achieve the desired goal.

$$E\Diamond(Loaded = 1 \text{ and } c < t_{bound}) \tag{4.10}$$

Uppaal Verification, Simulation and Path Creation Given the models of the robots within the team and their environments, and the desired mission expressed in a temporal logic, MEMO gives as output the plan that includes the actions that each robot is supposed to perform synchronized with the other team-mates paying particular attention to the temporal aspects and specifications. MEMO parses the generated plans and sends the

actions to the robots that then execute them. To ensure mission satisfaction under strict temporal constraints, the robots of the team may have to move together through the map without collide among them.

Figure 4.13 shows our tool dealing with a team of two robots, for each agent the tool computes a trajectory, in this case a and b, that each robot is supposed to follow in order to achieve the common mission. The commands of the two robots consider explicit time concerns in order to synchronize and meet in the same position at the same time.

Chapter 5

Implementation

In this chapter we analyse in details how our tool is implemented, Section 5.1 presents the scripts used in the automated process of the creation of the model. Section 5.2 shows the structure of the xml file in which the complete model of the system (robots and environment) is saved. Section 5.3 describes the main ideas behind the algorithms used to translate the environment into an equivalent Timed Automaton. Section 5.4 presents the procedure that enables the implementation of the results of the planner on the software simulator and the real robots.

5.1 Script General Description

In this section we will describe the different modules containing the algorithms upon which MEMO is based. We will focus on the creation of the XML file describing the behavior of the system (robots and environment). Specifically, MEMO is build upon the following scripts:

- **System2xml.py**: it provides the declarations and organization of xml file, it calls the function that creates the environment and then the one the creates the robots, if we have more than one robot in the space, this script simply calls the function different times. Specifically, the *System2xml.py* uses the *Environment2xmlEncodingX.py* and *CreateRobot.py* scripts.

- **Environment2xmlEncodingX.py**: parameter X specifies which one of the two encodings must be used. Taking as inputs the map, the geometrical properties of the building and the sample parameters creates the matrix that represent the map. It calls the function that extracts

from the figure.png the pixel coordinates of the walls, rooms and corridors, i.e., it calls the script *Figure2wall.py*

- **Figure2wall.py**: this function exclusively takes as input the image.png of the map and finds all the coordinates in pixel of the obstacles. The output is saved in a long bidimentional array which will be analysed by another script in order to organize these points in an efficient way.

- **CreateRobot.py**: this script is the most customizable of the system, it creates the model of the robot, the actions are synchronized with the ones of the model of the environment. The user has the possibility to choose between different standard models and also to modify it with his own ideas and specifications.

5.2 XML structure of a Timed Automata System

These algorithms are implemented in order to automatize the process of creation of the timed automata of the complete model of the system. The main element of the file.xml that describe the complete model are: declarations, template, locations, initial location, transition, system. In the following we explain the main characteristics of each single attribute.

```
<declaration>
    chan r,l,u,d,s;
    int t=0; int tmove=0;
    ...
</declaration>

<template>
    <name>Room</name>
    <location id="id0-0" x="0" y="0"></location>
    <init ref="id5-5" />
    <transition>
        <source ref="id0-1" />
        <target ref="id0-2" />
        <label kind="synchronisation">r?</label>
    </transition>
</template>

<template>
```

```
    <name>Robot</name>
    <location id="10" x="100" y="0" />
    <init ref="11" />
    <transition>
        <source ref="11" />
        <target ref="10" />
        <label kind="synchronisation">u!</label>
        <label kind="assignment">t=t+tmove</label>
    </transition>
</template>

<system>
    Room1=Room();
    Robot1=Robot();
    system Robot1, Room1;
</system>
```

- **DECLARATIONS**: At the beginning of every model we are supposed to define some parameters that we are going to use in the system. We must define all the names associated with the transitions and with the synchronizations, to define theme we use the term "Chan". It is also mandatory to define all the variables utilized, in our work we mainly deal with integer or boolean variables. It is possible to define global or internal variables. Global variables first are defined at the beginning and they are visible in all the system. Internal variables are defined in the sub-system of the system. For example, if an integer variable is defined inside the environment declarations it can only be seen and modified by actions and assignments of the relative sub-part of the model.

  Another type of parameters that are defined here are the clocks. A clock is responsible of keeping track of the time during the evolution of the system. In our work we define one global clock in order to be aware at what time each action or movement is performed and then we also define an internal clock for every robot in the environment, this is very useful to set the programmed duration of each activity of the robot. We need this additional clock because at each action we must set it to zero in order to keep the synchronization between the models.

- **TEMPLATE**: In our work we have two different types of template

describing the environment and the robot. The complete model of the system is composed by the combination of these two templates. For the environment, as previously explained, we implemented two kind of sub-template, namely encoding 1 and encoding 2, while for the robot we have several customizable models depending on their properties. Thanks to this modularity the user has the opportunity to create different combinations of elements and choose the one that best fits his specifications. Each template has a label in which is mandatory to define the name, in this work we selected "Environment" and "Robot", the names are really important for the final declaration of the complete system. A template consists in a set of states and transitions and successively we will explain in details how they are expressed in the file.xml .

- **LOCATION**: For each of these elements it is generated a state in the timed automata system. Each of them is characterized by an identifier, usually it is a sequential number or the subscripts of the element of relative matrix. These identifiers are really important for the specifications of the transitions. Another characteristic of this label is a pair of variables that represent the spacial coordinates of the state created in the UPPAAL environment of simulation. Setting these two variables is not mandatory, but by using them we are able to organise the states and and have proper graphical representation of the map during the simulation and it greatly simplifies the work of verification of the algorithm.

- **INITIAL LOCATION**: For each template it is mandatory to define from which state the execution of the actions starts. We set the initial point just writing one of the identifier of the states. Without this information we are not able to perform any simulation on the system. In our case, usually working on a system with two template, we must set an initial location for each template, paying particular attention to the synchronization between them, indeed performing actions, we will move from one state to another, so the starting configuration must be carefully chosen.

- **TRANSITION**: It creates a connection between two states, and it has a specified direction. Performing an action, the current properties of the system change. In order to express all the characteristics of a transition in a file.xml we must define five different labels: source, target, guard, assignment, synchronization.

Source and target must be defined for each transition, while the others are optional, they are defined to represent particular behaviours of the system. Transitions are a key point of a timed automata model, they describe and characterize the different options and evolutions of the model. For these reasons, while developing our algorithm that automatically creates states and transitions we must pay particular attention to this kind of element, an error in the definition of a transition could totally change the expected behaviour of the system and after that, find the mistake through tests and verifications might result very difficult and complicated. In UPPAAL Model Checker a transition is represented by an arrow that connects two states, in order to define it in file.xml we utilize the labels "source" and "target". To each label we assign a state identifier, and the simply represent the initial and final points of the arrow. We now describe the optional labels of the transitions.

The element "guard" indicates a condition "if" on the transition, if the condition is verified the transition is enabled and it is possible to reach the state define as a "target", otherwise if the condition is not verified the transition is cut and it is removed from the list of the possible actions that could be performed in that moment with that configuration of the system. We express these conditions as logic formula using the normal logic operators as "and", "or", "not" to formulate the desired relationships between the global and internal variables of the selected templates. In the templates of the robots we will make a great usage of these "guards" to express the duration (in real time) of the movements and actions performed by the robot. It works with the internal clock of the robot but, as a consequence of the synchronization between robot and environment it also works as a timer for the transition on the map. On the other hand we implement this label also in the xml model of the environment, in the "logic model", indeed with long strings of conditions we define all the physical constraints of the map, each transition is enabled if and only if between the source and target configuration there is not an obstacle. Later on we will explain in details how to automatically extrapolate all these conditions from the map.

The second xml label of the transition element is the "assignment", its function is to update or increment variables of the system after performing the relative transition. In our algorithms we generally use this function to reset a clock after a transition, to update the spacial

coordinates after a displacement or to set a boolean variable to one after the accomplishment of a sequence of actions.

Finally, the last label is the "synchronization", it is possible to name the transitions in two different ways, one adding an exclamation point or a question mark at the end of the name. This functions allows the programmer to put in relation transitions of different templates, when an action with the exclamation point is performed it follows the execution of the correspondent action in another template defined with the question mark. For example, when the robot realizes a movement, we witness also a change in the automata model of the map.

With these five xml labels is possible to completely define a timed automata transition, in our algorithms we will use these properties in order to automatically create a model that describe the real case in the best way possible, according to the user's requests and specifications.

- **SYSTEM**: The last part of the file.xml consists in the declarations of the templates selected, here we must specify which kind of model of the environment we are taking into account and also the type of robot. In this section we create the desired combination of the complete environment with the selected templates.

5.3 Algorithms

This section describes the main ideas used in the algorithms that are used to encode the environment into a TA. Sections 5.3.1 and 5.3.2 describes the algorithm used in encoding 1 and 2, respectively.

5.3.1 Encoding 1

The inputs and outputs of the Encoding 1 algorithm are described in the following:

- Input: Image.png of the map, array with the points of interest, real measure of the building, sampling parameter, diameter of the robot.
- Output: Xml file describing the Timed automata model of the environment.

With the encoding1 algorithm, the main idea is to create a direct and graphical correspondence between the map and the Timed Automata. According to the sampling parameter we perform a discretization of the map, we consider only some points of the image. We aim to describe the environment

through large matrix of automata states linked together in a proper way in order to describe the constraints on the movements defined by the walls. The steps describing the adopted algorithm are:

1. **Detecting the locations of walls.** The object "figure2wall", thanks to the libraries "cv2" and "numpy", calls the function cv2.imread(), cv2.cvtColor(), cv2.findCountours() and cv2.contourArea(). Given the image of the map we obtain as output a bidimentional array where all the coordinates of the wall are saved. These coordinates express the position (x, y) of every pixel of a wall on the map. The maximum values for x and y are determined by the pixel quality of the image, the origin is established in the upper left corner, the x axis goes from the origin to the right while the y axis goes towards the lower boundary of the image. Calling this function we extract the information of the wall from the map, basically our map is now described by this array. Despite the length of the array, the algorithm does not show any problem while working with walls described in this way.

2. **Moving from pixels to states.** In the second part of the algorithm we define all the parameters that depend on the primary inputs and can be calculated combining them. Using the map we only have measures in pixels, and it results impossible to combine the velocity of then robot with the displacements in the environment in order to compute the explicit time necessary to complete the task. To solve this problem initially we analyse the dimension of the image of the map, then we consider the input parameter $xmetersize$ that represents the real measure in meters of the building. Dividing the length of the picture in pixel by the measure in meter we obtain a scale factor (5.1)$[pixel/meter]$ that allows us to relate the image of the map with the real case of study.

$$\frac{imagedimension}{xmetersize} = scale \tag{5.1}$$

   We compute the equivalent values in pixels of the sampling parameter and of the diameter of the robot the were given in meter. All the coordinates of the points of interest were already set in pixel. We are supposed to define the specifications for the discretization of the map, as explained before we will represent the map as a large matrix. The ratio between the two sides of the matrix must be the same of the one between the two sides of the building, this is a key point of our work.

Using the scale parameter, from the measures in pixel of the map we compute the length in meter of the two sides of the building, then dividing these two values by the sampling parameter, which is expressed in meters, we obtain the total number of rows (5.3) and columns 5.2 where we will perform the discretiation, maintaining the ratio of the real measures.

$$\frac{horizontallenght}{sampling} = columns \tag{5.2}$$

$$\frac{verticallenght}{sampling} = rows \tag{5.3}$$

3. **Creating states of the TA.** In order to simplify the model we assume as hypothesis that our robot is programmed to move and perform actions inside of the building, for this reason, we would like to consider and discretize only the points inside the plan. In order to implement this simplification we colour in black the external part of the map, we could consider it as "the garden" of the building, and we take into account only the non-black pixels of the image. This modification will not cause any problem to the extraction of the coordinates of the walls. Generally this kind of robots work in an indoor environment, for this reason we consider this hypothesis to simplify our model, in the event that the programmer is also interested in the outdoor part of the map the algorithm will perfectly work creating a larger model.

   With two "for" cycles, the external one considering the number of the rows of the matrix and the internal one considering the columns, we aim to create the timed automata states matrix. Each state has an identifier that represents the two coordinates $(row, column)$ of the element of the matrix. In order to match each of these states with a point in pixel on the map (5.5), we multiply the coordinates of the identifier (that respectively goes from 0 to rows and from 0 to columns) for the sampling parameter expressed in pixel called factor (5.4)

$$sampling \times scale = factor \tag{5.4}$$

$$identifier \times factor = pixelcoordinate \tag{5.5}$$

   By doing this we create an association between each point of the states automata matrix and a point of the map expressed in pixel. With this

procedure we perform a sort of discretization of the image and now on we will consider only these sampled points. Following this algorithm multiplying the last element of the matrix for the factor parameter we will obtain the last point of the image in the lower right corner.

For each element of this cycle, if the correspondent discretized element on the image is not black we create a timed automata state, defining its location and name on the file.xml as shown in the previous paragraph. If we opened the file.xml with UPPAAL program we would obtain a set of circles/states organized in rows and columns, each one with its own identifier, with the external shape of the building took into account.

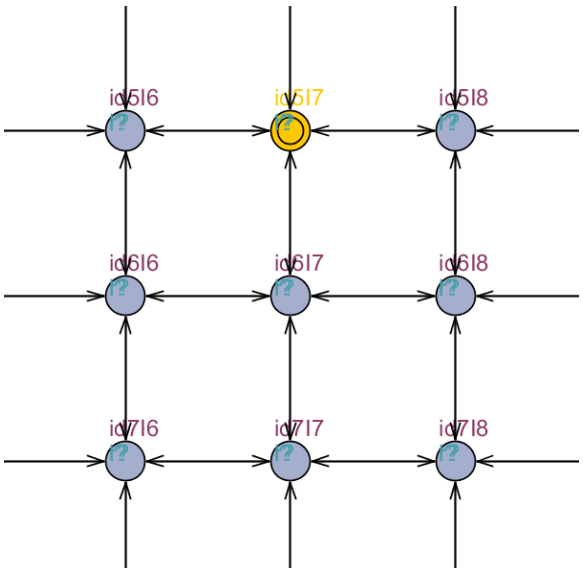

Figure 5.1: Initial state of the Timed Automata model.

4. **Initial state.** A mandatory characteristic to set in every timed automata model is the initial state, showed in Figure 5.1. The user is able to establish the initial location of the robot in the environment, in our work we though it could be a good idea to set the initial location in a room that will represent the "robot storage closet", where the machine stays during standby moments and where it could also charge its batteries. The user selects this location using pixel coordinates on the map. In this part of the algorithm we must find the correspondent state associated with the selected point on the map. Since the states are organized as matrix elements, usually a randomly chosen point is located inside the square defined by four states. In order to find the

closest state to the point we parse all the discritized point of the map. If the coordinate of the initial point is inside the square centred in the point-state, with side equal to the factor parameter (sampling parameter expressed in pixel), it means that the state is the closest one and we set the considered state as initial position. In order to set the initial location in the timed automata model we place the identifier of the found state into the relative label.

5. **Creating transitions.** Transitions are created to link the states of the timed automata model. In the algorithm by linking states with transitions we describe the characteristics of the map such as walls, rooms and corridors. We analyse all the combinations of consecutive states. Considering an example of two consecutive states, thanks to an implemented function we are able to check if between the correspondent discretized points on the map there is or there is not an obstacle that does not allow the robot to directly drive from the first point to the second. If there is not an obstacle along the direct path between the two point on the map, by writing the correct label on the file.xml, we create the transition in the timed automata model. For each pair of consecutive states linked by a transition we are supposed to create an element on the file.xml properly specifying two sub-labels. The first one called "source" is the point where the displacement starts while the "target" is the final point. We set these two labels writing the identifier of the two correspondent states. In this algorithm, in the case of the creation of the transition we must set other two labels of the file.xml, one the controls the synchronization with the actions of the robot and the other one that manages the assignments of the variables when the robot reaches or leaves a point of interest. In this algorithm it is not necessary to set guards or other constraints on the transitions. Indeed, in order to describe the the characteristics of the map we entirely rely on the presence or absence of the link between two consecutive states.

   The main idea of the algorithm is to parse all the descritized points on the map that we are going to consider depending on the sampling parameter, then for each of these elements we take into account possible 5 transition that express the directions of the possible movements of the robot. In this thesis for the sake of simplicity we work under the hypothesis that the robot is able to move in directions "perpendicular" to the walls, so the considered possible actions are Right, Left, Up, Down and Stay. These actions represent the directions with re-

spect to the absolute reference system of the room, they are not the relative actions of the robot. They could be seen as cardinal points, where for example the actions Right describe a displacement towards "Est" or Up towards "Nord". These method could be easily used also with diagonal transitions incrementing the complexity of the model. This kind of notation will greatly simplify the algorithm and the synchronization with the model of the robot. Considering each element of the matrix in two "for" cycle, we check the four states around the point and we call the implemented function able to detect the presence of obstacles.

6. **Checking for the presence of walls.** The function called "isThereWall" takes as input the long array that describe the walls of the map, the current state that we are considering, called source point and the consecutive state that we would like to connect with a transition, called target point. For each state we call this function once, in the function we implemented 4 parts, one for every direction of movements. The part of the algorithm that creates the action Stay does not need the call of this function, it just define a self-loop on the current state. Initially we set a boolean array with for elements, every element is set to zero and they respectively represent the 4 actions Right, Left, Down, Up. The function checks the presence of obstacles along the direction (as showed in Figure 5.2) and if the movement is not allowed in the map it sets to 1 the correspondent element and return the entire array. By looking to the elements set to one we are able to find the presence of walls. After the call of this function, for each elements we link the source point only with targets with the relative element equal to zero. For example, given a random point with a wall at its right, the returned array will be [1 0 0 0].

   In order to parse all the elements all the matrix we use two "for" cycles on rows, represented by the parameter "a" and on columns, represented by parameter "b". Then as in the the procedure of the creation of the states described above, we multiply a and b for the parameter factor and we obtain the correspondent coordinates in pixels of the point. The function isTherewall works with coordinates in pixel while the identifier of the states are expressed by the subscript of the matrix. This notation results very convenient in order to express source and target elements. Indeed, considering the two states linked by the transition Right, the target is identified by the pair (a,b) while to express the identifier of the target we simply increase the parameter

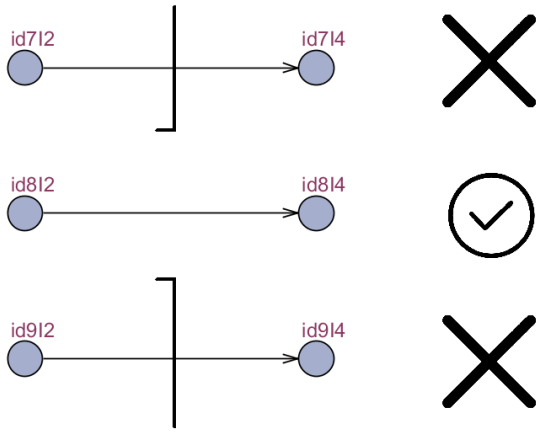

Figure 5.2: Transition creation strategy through a door of the building.

b, and the result is $(a, b+1)$. After this it is easy to map these two elements on the map with pixels coordinates and call the function. Considering another example with the action Up, the two identifiers will be (a, b) for the source and $(a-1, b)$ for the target. Analysing all the elements of the matrix, the source is always the current state and increasing or decreasing the coordinates of the source we consider the elements around.

During the conversion from the identifier of the elements of the matrix and the coordinates in pixel we must pay attention to a little difference of the two systems of reference. In fact, the first one, as elements of a matrix (a, b) the first parameter a represents the row while the second one represents the columns. On the other hand in the image of the maps the convention to describe the position of a pixel is with spacial coordinate (x, y), where the x axis corresponds to the horizontal direction and y axis the vertical one. For this reason we map the a parameter of the rows with the element y and analogously the b parameter of the columns with element x.

"How does the function isThereWall detect the presence of an obstacle along the direct path between two consecutive states?" In order to explain this we consider as example the transition Right and two random points:

- Source (a, b)
- Target $(a, b+1)$

The main idea is to check if in the area occupied by the robot during the complete movements and in the final configuration there is a wall. This is the reason why we must include in the algorithm of the environment parameters of the robot such the diameter from which we derive the radius. The area of the map in question is a rectangle, the apexes of the rectangle have the coordinates:

- $(xcoordinateofthesource)$
- $(xcoordinateofthethetarget) + radius$
- $(ycoordinateofthesource) + radius$
- $(ycoordinateofthesource) - radius$

in this particular case the horizontal side goes from the (x coordinate of the source) to the (x coordinate of the the target) + the radius of the robot while the vertical side of the rectangle has the same measure of the diameter and it is centered in the y coordinate of the source.

The area showed in Figure 5.3 covers the space occupied by the robot during the motion and also the space taken after the stop. It is very important to make the last consideration in order to avoid any kind of collision with the walls. We do not take into account the area occupied by the robot before the movement because it has been check in the previous iteration of the algorithm. We apply the same procedure for all the other directions, of course the rectangle into account will have a different position and in the case of actions Up and Down also a different orientation but the measures and the total area will be the same.

7. **Handling synchronization with the model of the robot.** After the analysis of the obstacles and after the decision of creating a transition we are supposed to set two other parameters, described by two different labels of the timed automata model which are the "synchronization" and the "assignment". The first one is used to create a connection between the model of the environment and the model of the machine, indeed we have a similar label also in the timed automata of the robot. Since the robot decides where to go and which actions wants to perform there is a dependency between the two models, the robot stands in an higher level with respect to environment. In order to express this behaviour of the complete model we use the same names of the action for both of the models with a little difference. The actions of the robot are followed by an exclamation point, while the

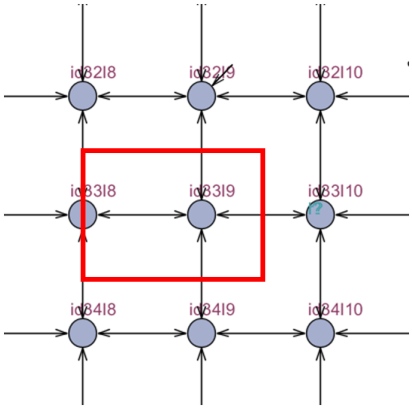

Figure 5.3: Area where the algorithm checks if there is an obstacles before creating a RIGHT transition.

same action in the environment is followed by a question mark. For example, if the robot decides to perform the action "r!", therefore the environment modifies its configuration performing action "r?". In our algorithm we implement this property to each transition created, in order to assign the correct label to the relative transition we use as before the parameters source and target. Depending on the coordinates of the target with respect to the source we set the synchronization label. For example, if the source is placed in coordinates (a, b) and the target in $(a + 1, b)$, the written label in the environment will be "u?" synchronized with the "u!" action of the robot. Analogously with coordinates $(a, b - 1)$ of the target the proper label will be "l?" synchronized with "l!". If we are in the case of the Stay action, where source and target are the same (a, b) we set the label "s?".

8. **Handling the points of interests.** Programming the second parameter of each transition results a bit more complicated. In order to simulate the behaviour of the robot that performs sequences of action in determined places of the environment, the user is supposed to choose these points on the map and give them as an input to the algorithm in the form of pixels coordinates (x, y). For each of these points the script associates a boolean variable set to zero, called Pn, where n is an integer number that goes from one to the number of selected points. $P1$ is the point where the robot is placed in the initial configuration of the system. To each of these points is associated a different action and the behaviour of the robot and the description of the sequences are described in the robot model, which will be described later. The

goal of this part of the algorithm is to identify the transitions that reach these points of interests and program the relative label to assign to right variable P to one in the case of reaching the point, and reset it to zero when leaving the point. It is very important to reset the variable to zero when we leave to point of interest because the robot is programmed to perform some actions as load and unload an object only if it is in the predetermined position. With the same procedure that we adopted for the initial point, we execute the conversion from the pixel coordinates to the correspondent state of the matrix in the timed automata model. We parse every discretized point on the map and we consider the closest state to the point of interest and then we are able to find the relative state to take into account expressed with (a, b) coordinates in the matrix. For each transition that we call an implemented function called "*getPoi*".

The function *getPoi* takes as input the list of the point of interests and the coordinates of the target and gives as output an integer from -1 to the number of the points of interest. In this function with a for cycle on the list we check if one of the poi has the same coordinates of the target of the transition just created, if this is verified it returns the position of the found element that we call k, if not it returns -1. Going back to the main script, in the "assignment" label we set the variable Pk to one if we have found a match, or to zero otherwise.

With this procedure, from an image, given the parameters of the environment and some parameters of the robot we are able to create the timed automata of different buildings. The simplicity of the inputs of this algorithm makes it a user friendly tool for programmers and applicable to a large scale of locations.

The proposed algorithm analyses every pair of consecutive states along the considered directions and check if between them there is an obstacles that could not allow a safe movement of the robot from the initial to the final configuration. If along this path no possible collisions are detected the tool automatically creates a directional arch that links the two states and labels it depending on the direction of the displacement.

5.3.2 Encoding 2

The two encoding share the same inputs but they produce a different output, which is however equivalent in terms of usage within the MEMO framework. The Encoding 2 relies on four main steps: (i) execution of the function

"figure wall"; (ii) definition of the secondary parameters; (iii) creation of the TA states and; (iv) creation of the TA transitions. The first two steps will not be described in the following since they are based on the same functions presented for Encoding 1.

- **Single state.** Differently fro Encoding 1, the second model is described by one single state, which is also set as the initial state. The tool keeps track of the position of the robot using two variables describing the spatial coordinates.

- **Logic constraints on the transitions.** Differently from Encoding 1, the information about the environment is not described by the states of the model; it is rather described in the logical conditions specified on the transitions of the TA. For each direction the algorithm creates a self-loop programmed to update the relative coordinate when the action is executed. The critical point of the definition of the transition is the implementation of the guards. For each movement direction, the algorithm considers all the points of the environment, and saves the coordinates of the points in which the robot cannot do a step in the considered direction since a wall is present. For each movement direction a self-loop is added to the single state of the automaton. Each self-loop has a guard that forbids the transition to be executed when a robot cannot do a step in the considered direction.

The features of synchronization, finding the points of interests and all the minor aspects describing the model are performed as in Encoding 1.

5.4 Choregraph Simulation and TurtleBot Commands

To convert the trace obtained from the UPPAAL model checker into the sequence of actions that must be executed by the robots, an appropriate procedure was executed. The trace encodes the points of interest as spatial coordinates (x, y) in the global reference of the system, which has the origin in the upper left corner of the environment and the two axis along the horizontal and vertical directions. However, the map of the robots had a different reference system. For this reason, regarding point reachability, the spatial coordinates were translated by means of a translation of the reference system into the coordinates of the robots. The trace encoded the actions that the robot had to perform. However, it was necessary to design an actual

mapping between the actions present in the model of the system (robots and their environments) and the functions that force the robot to execute specific actions. The designed mapping allowed calling specific functions that allow the execution of the desired action on the actual robot, when specific strings were parsed from the traces.

Chapter 6

Evaluation

This chapter evaluates MEMO. In Section 6.1 we evaluate the behavior of MEMO by considering different environments and robotic applications made by single and multiple robots. In Section 6.2 we evaluate the generated plans by using the Choregraph simulator. In Section 6.3 we show how the generated plans can be used in settings obtained by considering real environments.

6.1 Evaluating the planner

In this section we evaluate the ability of MEMO to support plan synthesis in real case scenarios. Our goal is to understand if MEMO is able to work by considering real maps. To evaluate how MEMO supports synthesis in real case scenarios we consider different maps, teams of robots, and missions, and we check how MEMO allows computing plans. We first considered a team of a single robot ($R1$) and then we evaluate how MEMO behaves with multiple robots ($R2$).

6.1.1 Team of a single robot

We analyze how MEMO behaves by considering different maps, missions and the two encodings described in Chapter 4.

- *Maps.* We consider three different buildings coming from real world scenarios:
  - the "Jupiter Building" of Chalmers University, Gothenburg, Sweden (Figure 6.1), which has approximately sizes of $80m \times 90m$ and in our experiments is described by the identifier $E1$.

– the “Edificio 22” of Politecnico di Milano (Figure 6.2), described by the identifier $E2$ with size of $120m \times 50m$.
– the “Edificio 20” of Politecnico di Milano (Figure 6.3), described by the identifier $E3$ with size of $70m \times 50m$.

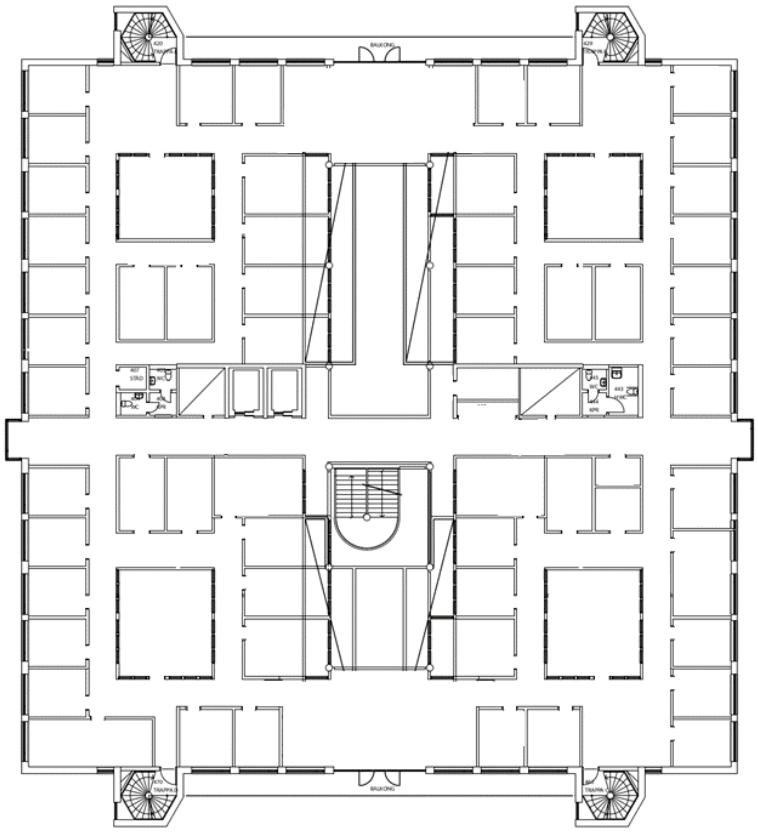

Figure 6.1: Jupiter Building, Chalmers University.

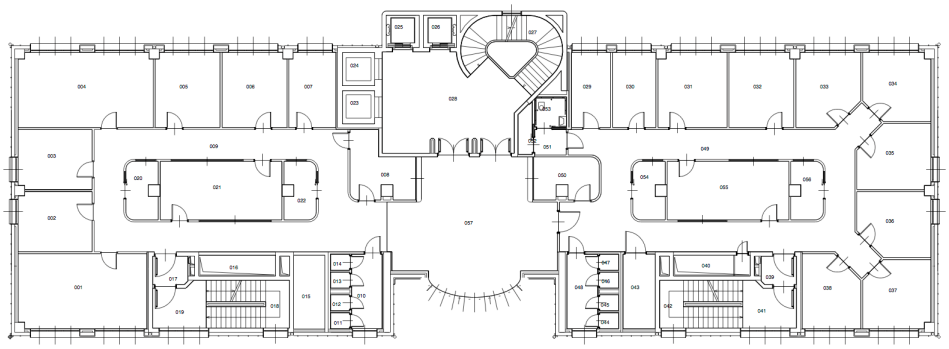

Figure 6.2: Edificio22, Politecnico di Milano.

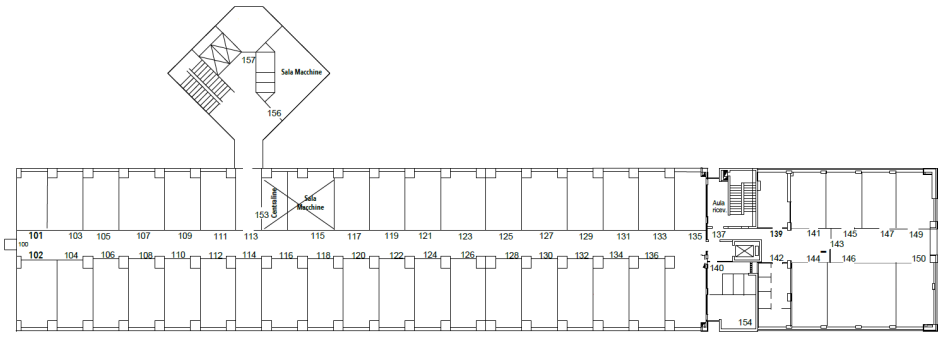

Figure 6.3: Edificio20, Politecnico di Milano.

In the phase of creating the Timed Automata model of the environment, for each map we discretize the space with four sampling parameters (ST): 50cm, 75cm, 100cm and 125cm.

- *Missions.* We considered three different missions with increasing complexity. The considered missions were based on the patterns for robotic missions presented in [31].
  - $M1$: this mission specifies the reachability of a point on the map. It checks if the robot, starting from the initial configuration of the system is able or not to reach the point selected by the user within the programmed time interval and according to the constraints described by the environment. For example, the mission "reach the coffee machine within 1 minute" is an example of mission $M1$.
  - $M2$: this mission takes into account the ability of the robot to move in the environment and perform actions. We verify the ability of the robot in performing a sequence of ordered actions. The specification consists of a sequence of two actions that should be performed in two points of the environment. Starting from the initial configuration the robot reaches the first point of interest and performs the first action, after waiting the amount of time needed for this task, the agent drives towards the second point of interest specified by the user where it is supposed to execute the second action. The characteristic of $M2$ is that these two points are defined in locations near the point where the robot starts the

	$M1$	$M2$	$M3$
$E1$	150s	500s	800s
$E2$	160s	300s	600s
$E3$	160s	300s	600s

Table 6.1: Values of $T1, T2$ and $T3$ for missions $M1, M2$ and $M3$.

movements. For example, the mission "reach the coffee machine, load the coffee and brings the coffee into the office 001 within 1 minute" is an example of mission $M2$.

– $M3$: this mission requires the robot to perform a sequence of two actions in two different locations. However, differently from $M2$ in $M3$ the selected points are in non-trivial locations with respect to the initial location of the robot. For example, the considered points may be at the opposite side of the building or in a room particularly difficult to reach. This specification aims to test also those cases in which finding a plan that satisfies the desired mission is not possible.

We considered each one of these missions over three different time interval $T1, T2$ and $T3$. The values of $T1, T2$ and $T3$ are different for each map, since the maps have different measures. The time intervals selected for the environments, considering the missions $M1, M2$ and $M3$ are presented in Table 6.1.

- We considered both the Encoding 1 and the Encoding 2, indicated in the Table 6.2 as $C1$ and $C2$, respectively.

Experimental setup. The models are generated considering different maps, different values for the sampling and different missions. We consider the team made by a single robot and we evaluated the three maps describing the environment $E1, E2$ and $E3$ previously described. This means that during the evaluation, for each map we create four environment models, each one obtained by considering a different value for the sampling parameter ST. This allows considering a different level of representation of the details. For every environment model created we perform the evaluation of the model considering the missions $M1, M2$ and $M3$. For each combination of environment, sampling parameter and mission we considered both the Encoding 1 (identified in the following as $C1$) and the Encoding 2 (identified in the following as $C2$). This leads to 216 different configurations summarized in Table 6.2.

		$M1$						$M2$						$M3$					
		$T1$		$T2$		$T3$		$T1$		$T2$		$T3$		$T1$		$T2$		$T3$	
E	**ST**	**C1**	**C2**	**C1**	**C2**	**C1**	**C2**	**C1**	**C2**	**C1**	**C2**	**C1**	**C2**	**C1**	**C2**	**C1**	**C2**	**C1**	**C2**
$E1$	50	39.9	8.7	37.2	8.2	44.6	8.2	40.2	2.5	41.6	2.7	40.8	2.7	101.2	93.4	61.0	32.5	61.5	32.4
$E1$	75	4.8	2.3	7.4	2.3	6.1	2.0	4.3	0.9	4.5	1.0	5.4	0.8	38.7	22.7	13.7	8.0	12.8	7.7
$E1$	100	2.4	0.9	1.9	1.0	1.9	1.0	1.9	0.5	1.7	0.5	1.5	1.0	17 .9	8.7	5.6	3.6	5.2	3.3
$E1$	125	0.8	0.5	1.0	0.7	0.8	0.5	0.6	0.3	0.7	0.3	0.7	0.3	8.5	4.9	3.4	1.8	2.8	1.8
$E2$	50	5.2	1.8	5.5	1.7	5.2	1.8	3.1	0.5	2.8	0.5	2.9	0.5	24.0	9.7	6.6	2.6	6.8	2.6
$E2$	75	4.5	1.1	4.7	1.1	4.5	1.3	2.4	0.5	2.4	0.4	2.6	0.4	20.6	5.0	2.0	0.7	6.5	1.7
$E2$	100	1.4	0.6	1.3	0.5	1.4	0.5	0.6	0.2	0.7	0.2	0.6	0.2	7.0	2.8	3.5	6.1	2.1	0.7
$E2$	125	0.5	0.2	0.5	0.1	0.5	0.6	2.5	0.2	2.5	0.1	2.5	0.2	3.5	0.2	0.8	0.1	0.8	0.3
$E3$	50	9.8	10.0	8.5	10.4	9.2	9.6	8.0	3.1	6.9	3.3	7.2	3.0	6.5	124.8	15.6	29.6	14.9	28.8
$E3$	75	2.5	1.7	2.3	1.5	2.4	1.5	2.0	0.6	1.5	0.6	1.6	0.6	6.2	30.8	4.8	5.5	4.8	5.8
$E3$	100	1.0	1.0	0.8	1.3	0.9	1.2	0.8	0.5	0.5	0.6	0.5	0.7	6.4	14.8	1.9	4.3	2.0	3.7
$E3$	125	0.3	0.5	0.5	0.2	0.5	0.2	4.3	0.3	0.3	0.3	0.3	0.3	3.4	6.7	1.0	1.5	1.2	1.5

Table 6.2: Results for experiment $EXP1$.
$E1$: "Jupiter Builing", $E2$: "Edificio22", $E3$: "Edificio20;
$M1$: Mission1, $M2$: Mission2, $M3$: Mission3; $C1$: Encoding1, $C2$: Encoding2;
$T1$: Time bound 1, $T2$: Time bound 2, $T3$: Time bound 3;
cells contain the time in seconds needed to compute a plan.

For each configuration we checked whether MEMO was able to produce a plan that ensured the mission satisfaction. If a plan was available we checked the correctness of the generated plan. In order to evaluate the usage of the procedure in real case scenarios, we recorded the time need to compute the plans.

Results. The time (in seconds) needed to compute plans is reported in the cells of Table 6.2. MEMO always succeeds in finding the right answer for the considered mission (plan available or not available). We noticed that the computational cost increased as the sampling value decreased, considering a given building. When the distance between points decreases, the number of points encoded in the TA increases. This aspect impacts on the time required from the planner. For encoding $C1$ the planning time required was at most 101.2 seconds, while for Encoding 2 is 124 seconds. These are reasonable times for a practical usage of the planner in real cases scenarios. In average, the time required for computing a plan and saving the trace was in the order of few seconds. In most of the cases Encoding 2 outperformed Encoding 1, and thus must be preferred for usage in real applications.

6.1.2 Team of two robots

We considered in details the behaviour of the planner while managing a team of two robots are moving and collaborating in the same environment.

- *Map.* We took into account the model of the environment $E1$ discretized every 100cm. We sampled the space with this value because it is a good compromise between the level of details described and computational cost required while dealing with two robots.

- *Missions.* We considered two missions the team of robots must achieve, which are based on the patterns for robotic missions presented in [31].

  - $M4$. Two robots start from two different initial locations, perform a path in order to meet at the same point of interest (one close to the other, without violating the constraints of collision avoidance) at the same time. Then, they synchronously execute a collaborative action. Finally, after performing the collaborative action the two robots are programmed to reach a destination point. This mission may for example be used for requiring "a robot to load an object on another robot in a particular location".
  - $M5$. It corresponds to mission $M4$, but the points where the

two robots are supposed to meet are in a non-trivial point with respect to the their initial positions.

We evaluated the mission considering two different time bounds and three meeting points $P1$, $P2$, and $P3$ with increasing distances w.r.t. to the initial positions of the robots.

Experimental setup. We considered the team made by a two robots, the map previously described and missions $M4$ and $M5$. We considered different time bounds. For each test we checked whether MEMO was able to produce a plan that ensured the mission satisfaction. If a plan was available we checked the correctness of the generated plan. In order to evaluate the usage of the procedure in real case scenarios, we recorded the time need to compute the plans. Since the computational cost of this experiment is elevated we set a time-out for the plan computation of 30 minutes.

Results. The approach succeeded in finding a plan for the team of two robots for mission $M4$ and $M5$ when points $P1$ and $P2$ were considered. When point P 3 was analysed the timeout was exceeded. The experiments show that the planning approach succeeds in considering a single robot even considering real maps. However, the approach does not scale when multiple robots must be handled due to state explosion problems. In this case, the usage of more complex planning procedures (maybe based on abstraction refinement) must be analysed.

6.2 Evaluating the plans on the Simulator

To check how the plans computed by MEMO can be used in practice, we first evaluated the produced plans by using the Choregraph simulator. We performed the following scenario. We considered environment $E1$, the Jupiter Building, and a team made by a single Nao robot. We assume that the robot had to perform the following missions, which are based on the patterns for robotic missions presented in [31]:

- $M6$: start from its initial location, reach a given position, get a set of items from a table that is present in the reached position, return to its initial location.
- $M7$: start from its initial location, reach a given position, say a sentence or a warning, return to its initial location
- $M8$: in its initial position the robot is loaded with an object by the user, reach a given position, unload the object, return to its initial position.

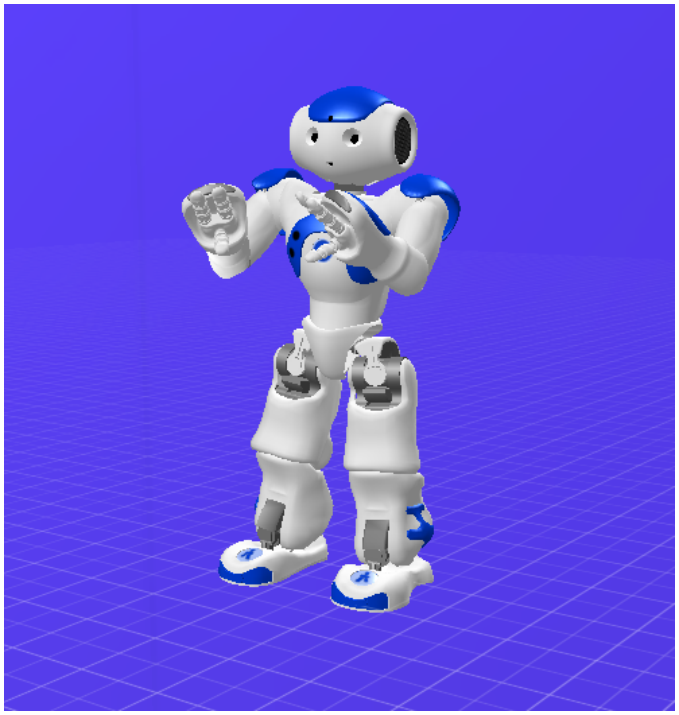

Figure 6.4: Nao robot, simulated in the Choregraph environment, lifting and object from a table after performing the plan compute by MEMO.

We have taken the plan generated by MEMO and send the actions to the Choregraph simulator. An overview of the simulator while performing the mission $M6$ can be seen in Figure 6.4.

We checked whether the actions where correctly performed by the robot.

The plans were successfully executed by the robot. We concluded that the approach proposed could be successfully integrated within real world robotic applications.

6.3 Experimental Evaluation

In order to check how MEMO can be used in practice, we use it in real case scenarios. We send the actions that were contained in the plans computed using MEMO to real robots and check that the robots behave as expected. We perform our experiments using the Turtlebot robot and the TIAGO robot. The experiments are conducted in the Jupiter Building of the Chalmers University and in the Pal Robotics Company.

Figure 6.5: TurtleBot delivering a cup of coffee to an office.

6.3.1 Experiment 1

We test the effectiveness of MEMO within the Jupiter Building of the Chalmers University. We consider three different scenarios in which the Turtlebot robot (Figure 6.5) has to achieve different missions.

Scenario 1. We assume that the Turtlebot is used as an indoor robot that is used to deliver coffees to the employee of the university. We assume the presence of a user at the coffee machine that is able to load a coffee cup on the Turtlebot. We considered the following mission.

Starting from the office (1), *reach the coffee machine* (2), *say that the user has* 10 *seconds to load the coffee cup on the robot, bring the coffee back to office* (1). *This mission must be accomplished within* 2 *minutes.*

We provide a map of the Jupiter Building of the Chalmers University obtained from the emergency fire plans of the building. We also provided a TA model of the Turtlebot. We gives these as inputs to MEMO. MEMO uses the map of the Jupiter Building to compute a TA that encoded the environment where the robot was deployed. MEMO uses the TA encoding

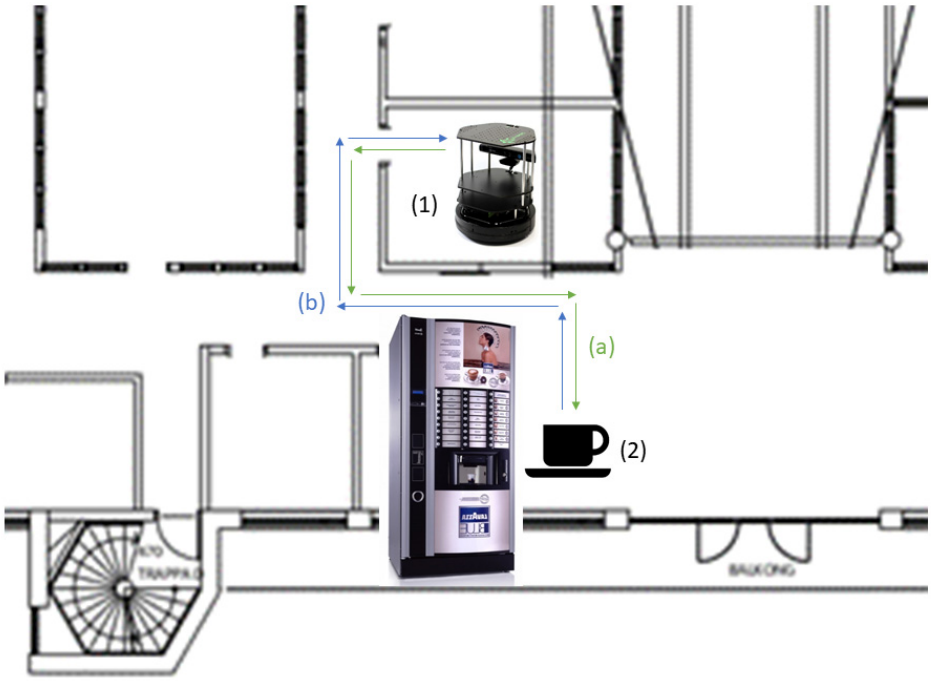

Figure 6.6: Mission 1: The robot delivers an hot cup of coffee in the office.

the robot and the TA of the environment to generate a plan that the robot has to follow. The plan is forwarded to the robot that successfully executes it. The plan is graphically described in Figure 6.6.

The robot starts from the office indicated in Figure 6.6 as (1) and follows the path indicated in Figure 6.6 as (a). The path allows the robot to reach the coffee machine (2). The robot then performs an action: it tells the user that he has 10 seconds to load the coffee and waits 10 seconds. Then it follows the path indicated in Figure 6.6 as (b) and finally reaches the office marked with symbol (1).

Scenario 2. We use the TurtleBot to perform a security mission. The robot has to detect the presence of an intruder in the building during the night. The robot while calling the surveillance reaches the point of the intrusion in order to record in details about the intrusion or to warn or detect the possible intruder.

Starting from the office (1)*, reach the point where the intrusion has been detected* (2) *within* 2 *minutes, record and check the situation, and finally go back to* (1)*.*

The robot starts from the office indicated in Figure 6.7 as (1) and follows the path indicated in Figure 6.7 as (a). The path allows the robot to reach

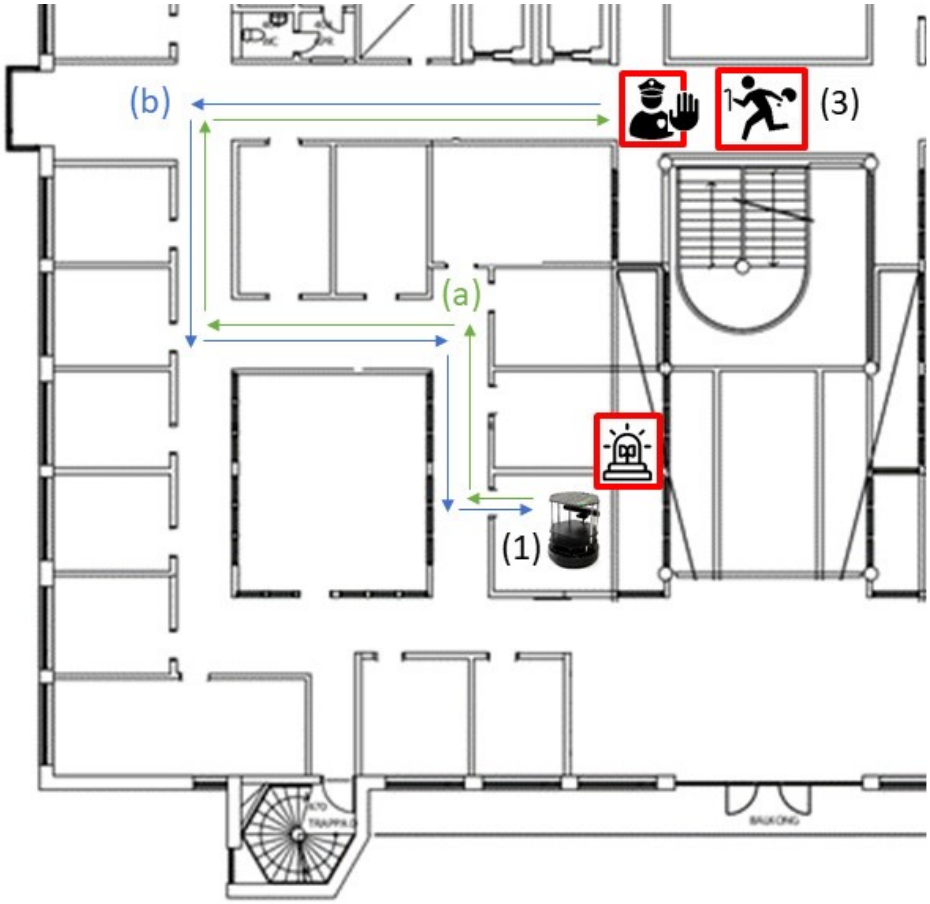

Figure 6.7: The robot detects and warns an intruder in the building.

the intrusion point (2). The robot then performs an action: record. Then it follows the path indicated in Figure 6.7 as (*b*) and finally reaches the office marked with symbol (1).

Scenario 3. The TurtleBot is exploited to automatically deliver a box or some documents from an office to another.

Starting from the office (1) *where the user loads the robot, reach the delivery point* (2) *within* 2 *minutes, wait the unloading, and finally go back to* (1).

The robot starts from the office indicated in Figure 6.8 as (1) and follows the path indicated in Figure 6.8 as (*a*). The path allows the robot to reach the delivery point (2). The robot then performs an action: wait the unloading. Then it follows the path indicated in Figure 6.8 as (*b*) and finally reaches the office marked with symbol (1).

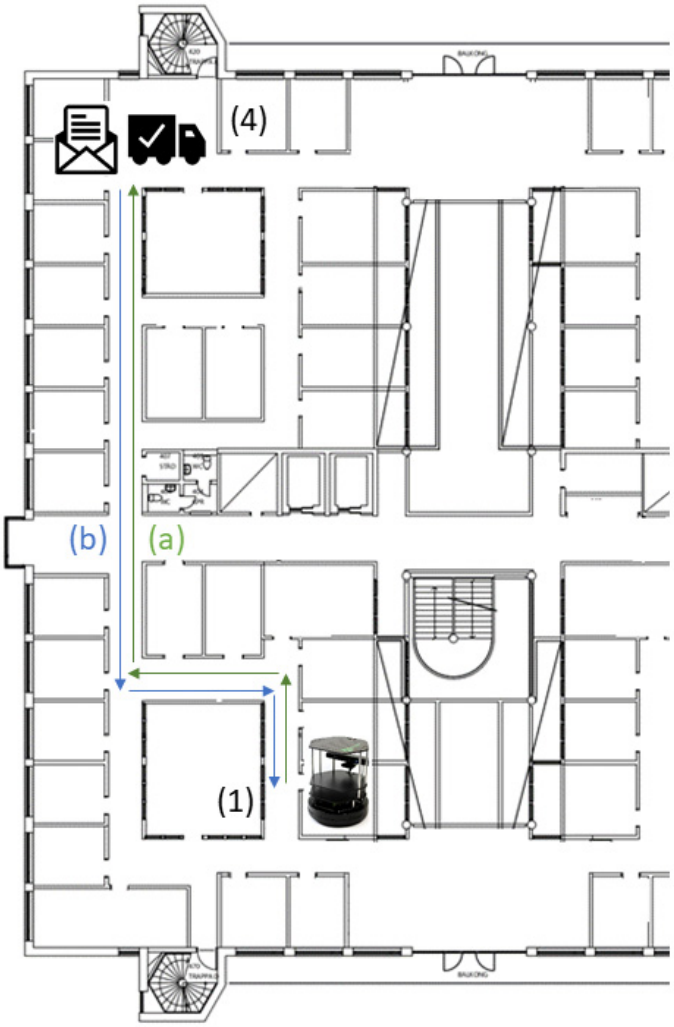

Figure 6.8: The robot delivers a box or a document from an office to another.

6.3.2 Experiment 2

We verify the effectiveness of MEMO in the offices of Pal robotics. We consider a simple scenario in which the TIAGO robot has to achieve a simple mission. We assume that the TIAGO robot has to reach a set of locations. Specifically, the following mission is considered.

Starting from location (1)*, reach location* (2)*, and then reach location* (3)*, go back to location* (1) *within* 3 *minutes.*

Since in this case the the maps of the buildings are not available, we manually design a Timed Automata and estimate the distances among locations (1), (2) and (3). We manually design a TA that encoded the behaviour

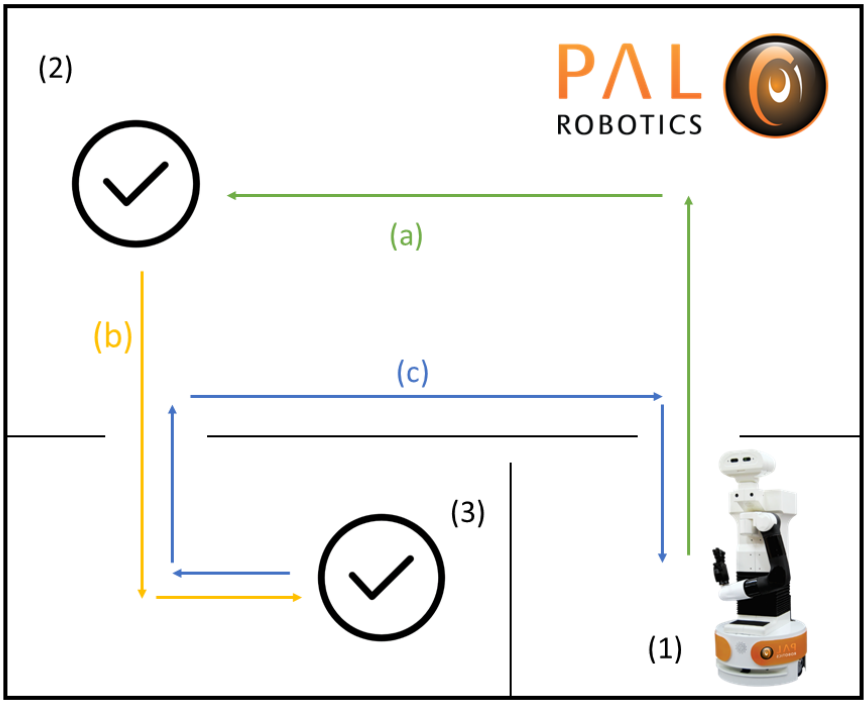

Figure 6.9: Scheme of the mission performed at Pal Robotics Company by TIAGO.

of the TIAGO robot. Then, we used MEMO to compute plans. MEMO successfully finds the desired plan. The plan is forwarded to the robot that successfully execute it. The plan is graphically described in Figure 6.9 and 6.10.

The robot starts from the room indicated in Figure 6.9 as (1) and follows the path indicated in Figure 6.9 as (a). The path allows the robot to reach the the point (2). Following the trajectory indicated in Figure 6.9 as (b) the robot reaches the point 3. Finally along the path (c) the robot goes back to the initial position.

6.3.3 Discussion

This section had evaluated MEMO considering three main aspects: (i) usage of the tool in different environments, with different teams of robots; (ii) usage of the plan in a simulated environment; and (iii) usage of MEMO in real world applications.

- The results showed that MEMO can be successfully used in different environments with a team composed by a single robot. When multiple robots are considered, the approach does not scale due to state

Figure 6.10: TIAGO Robot driving towards the desired location.

explosion problems. In this case, the usage of more complex planning procedures (maybe based on abstraction refinement) must be analyzed;

- The evaluation performed using the simulator showed that the plans can be successfully used in real world applications. That paved the way for a real world experimentation;
- The usage of MEMO in real world applications, showed that the computed plans can be successfully used in real cases. However, we still believe that there are some minor issues that still have to be addressed. In the following, we discuss the main ones
  - mismatch between the considered map and the real map of the building. Even if the considered maps were really good approximations of the environments in which the robots were deployed, the approach is still not fully automatic. There is still some manual tuning involved, which include: (i) the setting of the initial position of the robot, (ii) the alignment of the TA representing the map obtained from the figure and the one that is "learned" by the robot (e.g., the one learned using rviz [6]).
  - mismatch between the model of the robot and the actual behavior of the robot. We experience some problem regarding the behavior

of the robot. In some cases, the model of the robot was not perfectly accurate. For example, in some circumstances, to turn 90 degrees right the robot was actually do a complete circle to localize itself and then it was performing a 90 degrees turn.

We believe that the approach succeeded in providing good initial results on the usage of MEMO in real world applications. The problems described may be addressed in future works, where the analysis will not be a preliminary analysis of the overall framework (as the one described in this Chapter), but will be more focused on the actual support provided to the developers.

Chapter 7

Conclusions and future work

This thesis aimed at extending the support provided to developers in the creation of robotic applications. The analysis of the literature showed the absence of a general framework able to address the needs of the Co4robots [38] project, within which this thesis as been developed. Specifically, the analysis of the state of the art showed the absence of an approach able to address the planning problem when: (i) the missions of interest contain explicit time concerns; (ii) the team was composed by multiple robots; (iii) actions and synchronization among robots must be managed; (iv) the framework must work in realistic environments. This thesis developed a comprehensive approach (called MEMO) that supports developers in the creation of plans for the robots within a robotic application. Specifically, the contribution of this thesis can be summarized as follows:

- MEMO takes as inputs an image containing the map of the environment in which the robots will be deployed. For example, MEMO is able to process maps used in buildings to indicate fire emergency plans. The map of the buildings, together with some additional inputs provided by the user (i.e., the scale and the sampling parameters), are used by MEMO to detect geometrical measures and the general layout of the environment in which the robots will be deployed. Specifically, MEMO converts this map into a Timed Automaton (TA) that indicates how a robot can move within its environment, i.e., it encodes all the limitations of robot movements due to obstacles, walls, corridors and rooms. The creation of the TA model of the environment is based on the discretization of the space. We proposed two different ways to encode a map in a corresponding TA, indicated as Encoding 1 and Encoding 2.

- MEMO assumes that a TA model of each robot in the robotic team is provided as input. Since in the robotic world there exist several types of robots, with different abilities of movements, of performing actions and also different characteristics of the hardware, MEMO leaves to the user and robot manufacturers the possibility to customize and create new models for the robots of interest. This provides flexibility to the MEMO framework, and allows the usage of off-the-shelf models for the considered robots.

- MEMO is able to process missions containing explicit time concerns. Specifically, the mission is described using a Timed Computational Tree Logic (TCTL) specification. For example, MEMO is able to process missions such as "*the robot should reach the coffee machine, load the coffee and bring the coffee back within a time interval of 2 minutes*".

- MEMO combines the model on the environment, the one of the robot and the TCTL mission to compute plans that the robot should perform in order to achieve the satisfaction of the desired mission. MEMO relies on the UPPAAL Model Checker, a tool box for modeling, simulation and verification of real-time systems, based on constraint-solving and on-the-fly techniques. UPPAAL guarantees the possibility to deal with explicit time constraints, and is able to compute a trace that satisfies the TCTL mission of interest.

- the output trace produced by UPPAAL contains the points that the robot should visit in order to achieve the desired mission and the actions it should perform. We created and forwarded commands that were asking robots to reach locations and perform actions to the running robots.

- We evaluated MEMO considering three different aspects:

    1. we evaluated the behavior of MEMO by considering different environments and robotic applications considered by single and multiple robots. We combined different environments, inputs, parameters and types of missions[1] in order to check how MEMO supports developers in different situations. The results show that MEMO can be successfully used in different environments with a team composed by a single robot. When multiple robots are

[1]The considered missions were based on the patterns for robotic missions presented in [31].

considered, the approach due to state explosion problems. We also compare the efficiency of Encoding 1 and Encoding 2. The results showed that Encoding 2 must be preferred for usage in real robotic applications since in terms of computational cost it outperformed Encoding 1 in most of the cases.

2. we evaluated the generated plans by using the Choregraph simulator. The evaluation performed using the simulator showed that the plans can be successfully used in real world applications. That paved the way for a real world experimentation;
3. we show how the generated plans can be used in settings obtained by considering real environments. The real experiments were conducted in the "Jupiter Building" of Chalmers University and in the Pal Robotics Company (one of the partners of the project). In the experiments conducted in the "Jupiter Building" of Chalmers University, we considered the TurtleBot robot, a mobile platform able to move in narrow environments. We ask the TurtleBot robot to perform different missions: bring the coffee to the offices, perform security surveillance during the night and delivery objects in the building. In all the cases the approach successfully achieved its missions, (i) compute a plan from the map and the specification, (ii) deploy the right sequence of commands to the robot. In the experiments conducted in the Pal Robotics Company, we deployed our planner on the TIAGO Robot, a mobile platform of 145cm with a robotic arm. We considered different missions and a set of real case scenarios. The approach succeeded in all the experimental tests performed.

This work opens several directions of future work:

- improving the *implementation* of MEMO. At the moment, MEMO is still in a prototype version. We aim at providing a more stable implementation, that allows final users to use MEMO as off-the-shelf component. This includes fixing minor problems detected during the final experimentation in real world. During the experimentation, we had to perform some calibration of the tooling. This calibration mainly refer to mismatches between the considered map and the real map of the building. Even if the considered maps were really good approximations of the environments in which the robots were deployed, the approach is still not fully automatic. There is still some manual tuning involved, which include: (i) the setting of the initial position of

the robot, (ii) the alignment of the TA representing the map obtained from the figure and the one that is "learned" by the robot (e.g., the one learned using rviz [6]). We also aim to integrate MEMO within the Co4robots architecture [13].

- improving the *ability of the model to describe the real scenarios*. We worked under the hypothesis that the movements of the robot were limited to perpendicular directions. However, during the experimentation we observed mismatches between the model of the robot and its actual behavior. For example, in some circumstances, to turn 90 degrees right the robot was actually do a complete circle to localize itself and then perform a 90 degrees turn. More complex models may consider these behaviors and also allow modeling diagonal movement directions or considering particular kinds of points interpolation.

- improving *performances when teams of robots are considered*. As discussed in the evaluation performance issues were present when multiple robots were considered. These issues were caused by the well known state explosion problems. To solve these issues, the usage of more complex planning procedures (maybe based on abstraction refinement) must be investigated. For example, abstracting some information of the complete model of the environment would provide more compact TA models for the environment and provide benefits in the computational costs of the planning.

- considering *more complex models* that allow the description of different aspects of the robotic application within the planning procedure. These aspects may include modeling the robotic application using more reacher models, such as the one that support "Game Theory" reasoning techniques. In this case, the planning problem may be considered as a two player game where the two players are the robotic team of robots and their environment. Other aspects may include uncertainty about the models of the robotic application. Uncertainty has been deeply considered using multi-valued logics in the software engineering and formal methods communities (see for example [26, 28, 29, 30, 9]), but has mainly been addressed by using probabilistic models in the robotic field. The usage of multi-valued logics to encode uncertainty in the models of the robots is an intriguing research direction. A preliminary discussion about this topic can be found in [27].

Bibliography

[1] http://www.pal-robotics.com/en/home/.

[2] https://ifr.org/worldrobotics/.

[3] https://www.ald.softbankrobotics.com/en/robots/nao/find-out-more-about-nao.

[4] https://www.turtlebot.com/.

[5] http://tiago.pal-robotics.com/.

[6] http://wiki.ros.org/rviz.

[7] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[8] Michael S. Andersen, Rune S. Jensen, Thomas Bak, and Michael M. Quottrup. Motion planning in multi-robot systems using timed automata. *IFAC Proceedings Volumes*, 37(8):597 – 602, 2004. IFAC/EURON Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal, 5-7 July 2004.

[9] A. Bernasconi, C. Menghi, P. Spoletini, L. D Zuck, and C. Ghezzi. From model checking to a temporal proof for partial models: preliminary example. *arXiv preprint arXiv:1706.02701*, 2017.

[10] Y. Chen, X. C. Ding, and C. Belta. Synthesis of distributed control and communication schemes from global ltl specifications. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 2718–2723, Dec 2011.

[11] Y. Chen, J. Tůmová, and C. Belta. Ltl robot motion control based on automata learning of environmental dynamics. In *2012 IEEE International Conference on Robotics and Automation*, pages 5177–5182, May 2012.

[12] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas. Temporal logic motion planning for dynamic robots. *Automatica*, 45(2):343 – 352, 2009.

[13] S. Garcia, C. Menghi, P. Pelliccione, T. Berger, and R. Wohlrab. An architecture for decentralized, collaborative, and autonomous robots. In *International Conference on Software Architecture (ICSA), Seattle, USA*, 2018.

[14] M. Guo and D. V. Dimarogonas. Reconfiguration in motion planning of single- and multi-agent systems under infeasible local ltl specifications. In *52nd IEEE Conference on Decision and Control*, pages 2758–2763, Dec 2013.

[15] M. Guo, K. H. Johansson, and D. V. Dimarogonas. Motion and action planning under ltl specifications using navigation functions and action description language. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 240–245, Nov 2013.

[16] M. Guo, K. H. Johansson, and D. V. Dimarogonas. Revising motion planning under linear temporal logic specifications in partially known workspaces. In *ICRA*, pages 5025–5032. IEEE, 2013.

[17] M. Guo, J. Tůmová, and D. V. Dimarogonas. Cooperative decentralized multi-agent control under local ltl tasks and connectivity constraints. In *53rd IEEE Conference on Decision and Control*, pages 75–80, Dec 2014.

[18] M. Guo, J. Tůmová, and D. V. Dimarogonas. Hybrid control of multi-agent systems under local temporal tasks and relative-distance constraints. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 1701–1706, Dec 2015.

[19] T. Kehrer, C. Tsigkanos, and C. Ghezzi. An emof-compliant abstract syntax for bigraphs. In *Proceedings Second Graphs as Models Workshop, GaM@ETAPS 2016, Eindhoven, The Netherlands, April 2-3, 2016.*, pages 16–30, 2016.

[20] M. Kloetzer and C. Belta. Ltl planning for groups of robots. In *2006 IEEE International Conference on Networking, Sensing and Control*, pages 578–583, 2006.

[21] M. Kloetzer and C. Belta. Control of multi-robot teams based on ltl specifications. *IFAC Proceedings Volumes*, 40(18):103 – 108, 2007. 4th

IFAC Conference on Management and Control of Production and Logistics.

[22] M. Kloetzer and C. Belta. Automatic deployment of distributed teams of robots from temporal logic motion specifications. *IEEE Transactions on Robotics*, 26(1):48–61, Feb 2010.

[23] T. John Koo, RongQing Li, Michael M. Quottrup, Charles A. Clifton, Roozbeh Izadi-Zamanabadi, and Thomas Bak. A framework for multi-robot motion planning from temporal logic specifications. *Science China Information Sciences*, 55(7):1675–1692, Jul 2012.

[24] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, Nov 1990.

[25] Kim G Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International journal on software tools for technology transfer*, 1(1-2):134–152, 1997.

[26] C. Menghi. Verifying incomplete and evolving specifications. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 670–673. ACM, 2014.

[27] C. Menghi, S. Garcia, P. Pelliccione, and J. Tumova. Poster: Towards multi-robot applications planning under uncertainty. In *Companion Proceedings of the 40th International Conference on Software Engineering*, ICSE Companion 2018.

[28] C. Menghi, P. Spoletini, and C. Ghezzi. Dealing with incompleteness in automata-based model checking. In *FM 2016: Formal Methods - 21st International Symposium, Limassol, Cyprus, November 9-11, 2016, Proceedings*, volume 9995, pages 531–550. Springer, 2016.

[29] C. Menghi, P. Spoletini, and C. Ghezzi. Cover: Change-based goal verifier and reasoner. In *REFSQ Workshops*, 2017.

[30] C. Menghi, P. Spoletini, and C. Ghezzi. Integrating goal model analysis with iterative design. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 112–128. Springer, Cham, 2017.

[31] C. Menghi, C. Tsigkanos, T. Berger, P. Pelliccione, and C. Ghezzi. Poster: Property specification patterns for robotic missions. In *Companion Proceedings of the 40th International Conference on Software Engineering*, ICSE Companion 2018.

[32] A. Nikou, D. Boskos, J. Tůmová, and D. V. Dimarogonas. Cooperative planning for coupled multi-agent systems under timed temporal specifications. In *2017 American Control Conference (ACC)*, pages 1847–1852, May 2017.

[33] A. Nikou, J. Tůmová, and D. V. Dimarogonas. Cooperative task planning of multi-agent systems under timed temporal specifications. In *2016 American Control Conference (ACC)*, pages 7104–7109, July 2016.

[34] L. Pasquale, C. Ghezzi, C. Menghi, C. Tsigkanos, and B. Nuseibeh. Topology aware adaptive security. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 43–48. ACM, 2014.

[35] M. M. Quottrup, T. Bak, and R. I. Zamanabadi. Multi-robot planning: a timed automata approach. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 5, pages 4417–4422 Vol.5, April 2004.

[36] E. Rabiah and B. Belkhouche. Formal specification, refinement, and implementation of path planning. In *2016 12th International Conference on Innovations in Information Technology (IIT)*, pages 1–6, Nov 2016.

[37] S. L. Smith, J. Tůmová, C. Belta, and D. Rus. Optimal path planning under temporal logic constraints. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3288–3293, Oct 2010.

[38] The European Commission. *EU H2020 Research and Innovation Programme under GA No. 731869 (Co4Robots)*. The European Commission, 2016-2017.

[39] J. Tůmová, L. I. Reyes Castro, S. Karaman, E. Frazzoli, and D. Rus. Minimum-violation ltl planning with conflicting specifications. In *2013 American Control Conference*, pages 200–205, June 2013.

[40] J. Tůmová and D. V. Dimarogonas. Multi-agent planning under local ltl specifications and event-based synchronization. *Automatica*, 70:239 – 248, 2016.

[41] J. Tůmová, A. Marzinotto, D. V. Dimarogonas, and D. Kragic. Maximally satisfying ltl action planning. In *2014 IEEE/RSJ International*

Conference on Intelligent Robots and Systems, pages 1503–1510, Sept 2014.

[42] C. Tsigkanos, T. Kehrer, and C. Ghezzi. Architecting dynamic cyber-physical spaces. *Computing*, 98(10):1011–1040, 2016.

[43] C. Tsigkanos, T. Kehrer, and C. Ghezzi. Modeling and verification of evolving cyber-physical spaces. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, pages 38–48, 2017.

[44] C. Tsigkanos, T. Kehrer, C. Ghezzi, L. Pasquale, and B. Nuseibeh. Adding static and dynamic semantics to building information models. In *Proceedings of the 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems, SEsCPS@ICSE 2016, Austin, Texas, USA, May 14-22, 2016*, pages 1–7, 2016.

[45] C. Tsigkanos, L. Pasquale, C. Ghezzi, and B. Nuseibeh. On the interplay between cyber and physical spaces for adaptive security. *IEEE Transactions on Dependable and Secure Computing*, PP(99):1–1, 2016.

[46] C. Tsigkanos, L. Pasquale, C. Menghi, C. Ghezzi, and B. Nuseibeh. Engineering topology aware adaptive security: Preventing requirements violations at runtime. In *Requirements Engineering Conference (RE), 2014 IEEE 22nd International*, pages 203–212. IEEE, 2014.

[47] Y. Zhou, D. Maity, and J. S. Baras. Optimal mission planner with timed temporal logic constraints. In *2015 European Control Conference (ECC)*, pages 759–764, July 2015.