Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Μηχανολογικων Κατασκευων & Αυτοματου Ελεγχου

Σύνθεση Αναδιαμορφώσιμου Διακριτού Ελεγκτή για Ομάδα Ρομπότ σε Σχήμα Οδηγού-Ακολούθων με βάση Ατομικές Προδιαγραφές

Διπλωματικη Εργασια

του

διονυσιου θ. μανουσακα

Επιβλέπων: Κωνσταντίνος Ι. Κυριακόπουλος
Καθηγητής Ε.Μ.Π.

Εργαστηριο Αυτοματου Ελεγχου

Αθήνα, Ιούλιος 2014

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Μηχανολογικών Κατασκευών & Αυτομάτου Ελέγχου
Εργαστήριο Αυτομάτου Ελέγχου

Σύνθεση Αναδιαμορφώσιμου Διακριτού Ελεγκτή για Ομάδα Ρομπότ σε Σχήμα Οδηγού-Ακολούθων με βάση Ατομικές Προδιαγραφές

Διπλωματικη Εργασια

του

ΔΙΟΝΥΣΙΟΥ Θ. ΜΑΝΟΥΣΑΚΑ

Επιβλέπων: Κων/νος Κυριακόπουλος
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 19η Ιουλίου 2014.

.....	.....	.....
Κων/νος Κυριακόπουλος	Α.- Γ. Σταφυλοπάτης	Κων/νος Τζαφέστας
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2014

.....

Διονυσιος Θ. Μανουσακας
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Ευχαριστίες

Με την ανά χείρας διπλωματική εργασία ολοκληρώνονται οι προπτυχιακές σπουδές μου στη σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών του Εθνικού Μετσοβίου Πολυτεχνείου. Το περιεχόμενο της εργασίας αποτελεί τους καρπούς έρευνας που διεξήγαγα στο εργαστήριο Αυτομάτου Ελέγχου των Μηχανολόγων Μηχανικών υπό την εποπτεία του καθ. Κ. Κυριακόπουλου. Ευχαριστώ θερμά τον επιβλέποντα για την εμπιστοσύνη του και την ευκαιρία που μου παρείχε να εργαστώ στην ερευνητική του ομάδα, καθώς επίσης για τη μετάδοση της εμπειρίας και της διαίσθησής του στο αντικείμενο της εργασίας και την καθοδηγητική του ανάδραση στην πορεία εκπόνησής της. Ιδιαίτερες ευχαριστίες απευθύνω στον ερευνητή δρ. Χ.Μπεχλιούλη για τη στενή συνεργασία, τους συχνούς διαλόγους μας και τις διαφωτιστικές συμβουλές του. Εκφράζω επιπλέον τις ευχαριστίες μου προς τους καθ. Α. Σταφυλοπάτη και Ι. Τσινιά για όσα μου προσέφεραν διδακτικά με τα προπτυχιακά τους μαθήματα, καθώς και για την καθοριστική και έμπρακτη συμβολή τους στη διαδικασία των αιτήσεων για μεταπτυχιακές σπουδές. Ευχαριστώ επίσης όλα τα μέλη της ερευνητικής ομάδας του εργαστηρίου. Τέλος, θα ήταν παράλειψή μου να μην αναφερθώ στην ανεκτίμητη στήριξη της οικογένειάς μου σε όλη τη μακρά πορεία της φοίτησής μου.

Περίληψη

Μία συνήθης εργασία σε ένα σύστημα πολλών κινητών ρομπότ σε αρχιτεκτονική οδηγού - ακολούθων είναι η ανάθεση στον οδηγό μιας διαδρομής σε ένα (μερικώς) άγνωστο χώρο. Η διαδρομή αυτή προκύπτει από την προβολή στο χώρο εργασίας ενός συνόλου σύνθετων ενεργειών που πρέπει να εκτελέσει ο οδηγός, όπως η επίσκεψη περιοχών του χώρου εργασίας με συγκεκριμένη σειρά ή περιοδικότητα, που ως επί το πλείστον έχουν διατυπωθεί σε μια αρκούντως εκφραστική τυπική γλώσσα (π.χ. LTL). Αντίστοιχα, σε καθέναν από τους ακολούθους ανατίθεται να διατηρούν επικοινωνία με ένα σύνολο από άλλα ρομπότ εντός της ομάδας. Ένα μείζον ζητούμενο στα συστήματα πολλών ρομπότ είναι η συνεχής διατήρηση της ολικής συνδεσιμότητας του δικτύου. Ωστόσο, η κίνηση του συστήματος σε έναν άγνωστο χώρο εργασίας με ένα σχήμα όπως το ανωτέρω συνεπάγεται ότι για τη διέλευση του δικτύου από περιοχές με εμπόδια συχνά απαιτείται η αναπροσαρμογή των σχέσεων τοπικής συνδεσιμότητας μεταξύ των ρομπότ, ούτως ώστε η εργασία να περατώνεται (δηλαδή το σύστημα να κινείται στη δοθείσα διαδρομή) χωρίς απώλεια της ολικής συνδεσιμότητας.

Στην παρούσα εργασία προτείνουμε: (α) έναν κατανεμημένο αλγόριθμο που επιτρέπει την κίνηση του συστήματος με διατήρηση των σχέσεων τοπικής συνδεσιμότητας, (β) έναν αλγόριθμο δυναμικής αναδιαμόρφωσης των σχέσεων τοπικής συνδεσιμότητας που διατηρεί την ολική συνδεσιμότητα του δικτύου, όταν δεν επιτρέπεται η κίνηση του συστήματος με βάση το αρχικό σύνολο προδιαγραφών γειτνίασης λόγω εμποδίων στο χώρο εργασίας.

Λέξεις Κλειδιά

Σύστημα Πολλαπλών Ρομπότ, Σχεδίαση Διαδρομών / Κατανεμημένος Συντονισμός Κινητών Ρομπότ, Κατανεμημένοι Αλγόριθμοι, Σχήμα Οδηγού-Ακολούθων, Υβριδικά Συστήματα, Διατήρηση Ολικής Συνδεσιμότητας, Αναδιαμορφωσιμότητα, Κατανεμημένη Τεχνητή Νοημοσύνη, Αλγεβρική Θεωρία Γράφων, Πρόβλημα Ικανοποίησης Περιορισμών, Υπεπεριορισμένο Πρόβλημα, Ελάχιστα Μη-Ικανοποιήσιμοι Πυρήνες

Abstract

A common task for a leader- follower type multi-robotic system deployed in a (partially) unknown workspace is the assignment of a desired path to the leader. This path is usually derived from the projection to the workspace of a set of complicated mobility tasks which should be executed by the leader, such as visiting a sequence of regions with specific order, iterations or periodicity, and is formulated in a sufficiently expressive formal language (e.g. LTL). Respectively, each follower is assigned to keep contact with a subset of robots within the team. A major issue in multi-robotic systems is the constant maintenance of global connectivity of their underlying topology. However, system's motion in an unknown workspace in the above-described architecture implies the necessity for reconfiguration of local connectivity specifications (e.g. when robots pass through neighbourhoods of obstacles), so that task is being excuted (namely system moves on the path without getting stuck) and global connectivity is not violated.

In this thesis we propose: (a) a distributed algorithm which allows systems motion while preserves the set of local connectivity specifications whenever possible , (b) a (centralized) algorithm that dynamically reconfigures local connectivity specifications of robots without violating global connectivity, when the system based on the initial set of specifications gets stuck due to obstacles encountered in workspace.

Keywords

Multi-robotic system, Path Planning / Distributed Coordination of Mobile Robots, Distributed Algorithms, Hybrid Systems, Global Connectivity Maintenance, Reconfigurability, Leader-Follower Scheme, Distributed Artificial Intelligence, Algebraic Graph Theory, Constraint Satisfaction Problem(CSP), Overconstrained Problems, Minimal Unsatisfiable Cores (MUCs)

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

In this introductory chapter we cover recent research activity in multi-agent systems, with particular focus on the issues of multi-agent coordination, symbolic planning and global connectivity maintenance. The aim is to arm the reader with a concise, modern summary of fundamental notions and concepts that will be used later, while revealing achievements from the recent bibliography, difficulties and work to be done on the area.

1.1 Overview of Distributed Multi-agent Coordination

1.1.1 History

Distributed coordination of multiple robots, including unmanned aerial vehicles, unmanned ground vehicles and unmanned underwater robots, has been a very active research subject studied extensively by the systems and control community. The recent results in this area are categorized into several directions, such as consensus, formation control, optimization, task assignment, and estimation.

Control theory and practice may date back to the beginning of the last century when Wright Brothers attempted their first test flight in 1903. Since then, control theory has gradually gained popularity, receiving more and wider attention especially during the World War II when it was developed and applied to fire-control systems, missile navigation and guidance, as well as various electronic automation devices. In the past several decades, modern control theory was further advanced due to the booming of aerospace technology based on large-scale engineering systems.

1.1.2 Centralized vs. Decentralized Approach

During the rapid and sustained development of the modern control theory, technology for controlling a single robot, albeit higher-dimensional and complex, has become relatively mature and has produced many effective tools such as PID control, adaptive control, nonlinear control, intelligent control, predictive and robust control methodologies.

Nowadays network science has emerged as a powerful conceptual paradigm in science

and engineering [23]. Constructs and phenomena such as interconnected networks, random and small-world networks, and phase transition appear in a wide variety of research literature, ranging across social networks, statistical physics, sensor networks, economics, and of course multiagent coordination and control. The reason for this unprecedented attention to network science is twofold. On the one hand, in a number of disciplines - particularly in biological and material sciences - it has become vital to gain a deeper understanding of the role that inter-elemental interactions play in the collective functionality of multilayered systems. On the other hand, technological advances have facilitated an ability to synthesize networked engineering systems - such as those found in multivehicle systems, sensor networks, and nanostructures - that resemble, sometimes remotely, their natural counterparts in terms of their functional and operational complexity.

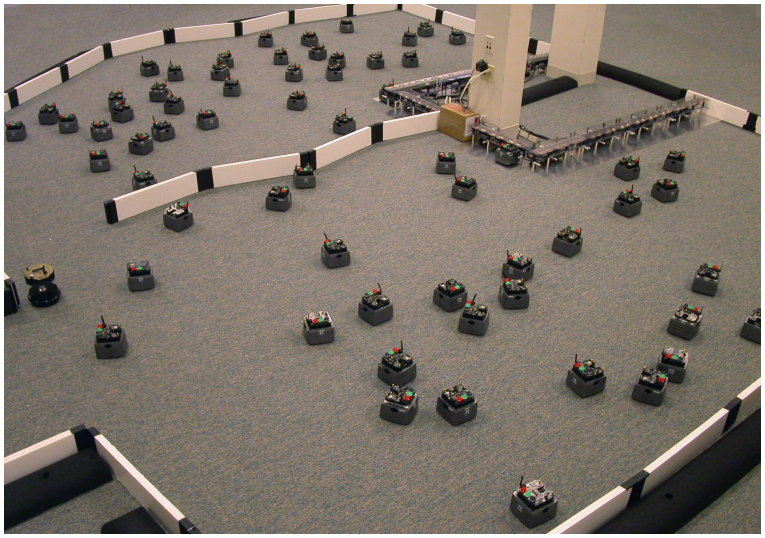


Figure 1.1: Swarm of mobile robots (from Rice Multi-Robot Systems Lab)

In the past two decades in particular, control of multiple robots has received increasing demands spurred by the fact that many benefits can be obtained when a single complicated robot is equivalently replaced by multiple yet simpler ones. In this endeavour, two approaches are commonly adopted for controlling multiple robots: a centralized approach and a distributed approach. The **centralized** approach is based on the assumption that a central station is available and powerful enough to control a whole group of robots. Essentially, the centralized approach is a direct extension of the traditional single-robot-based control philosophy and strategy. On the contrary, the **distributed** approach does not require a central station for control, at the cost of becoming far more complex in structure and organization.

A distributed system consists of a network of agent nodes, each with its own processing facility, which together do not require any central fusion, control or communication facility. In a distributed system, fusion and control occur locally at each node on the basis of local observations and the information communicated from neighbouring nodes, with no need for a common place where fusion or global decisions are made.

Although both approaches are considered practical depending on the situations and conditions of the real applications, the distributed approach is believed more promising due to many inevitable physical constraints such as limited resources and energy, short wireless communication ranges, narrow bandwidths, and large sizes of robots to manage and control. (For a more detailed presentation of the topic reader can refer to [6].)

In distributed control of a group of autonomous mobile robots, the main objective typically is to have the whole group of robots working in a cooperative fashion throughout a distributed protocol. Here, cooperative refers to a close relationship among all robots in the group where information sharing plays a central role.

Advantages of decentralization

The distributed approach has many advantages in achieving cooperative group performances, especially with low operational costs, less system requirements, high robustness, strong adaptivity, and flexible scalability, therefore has been widely recognized and appreciated.

In particular, a distributed scheme is characterised by three constraints ([16]):

1. There is no single central decision centre; no one node should be central to the successful operation of the network.

2. There is no common communication facility; nodes cannot broadcast results and communication must be kept on a strictly node-to-node basis.

3. Nodes do not have any global knowledge of network topology; nodes should only know about connections in their own neighbourhood.

The constraints imposed provide a number of important characteristics:

- Eliminating the central decision centre and any common communication facility ensures that the system is **scalable** as there are no limits imposed by centralised computational bottlenecks or lack of communication bandwidth.

- Ensuring that no node is central and that no global knowledge of the network topology is required for control means that the system can be made **survivable** to the on-line loss (or addition) of sensing nodes and to dynamic changes in the network structure.

- As all decision processes must take place locally at each site and no global knowledge of the network is required a priori, nodes can be constructed and programmed in a **modular** fashion.

Drawbacks of decentralization

Although decentralization shows obvious advantages over centralization, such as scalability and robustness, decentralization also has its own drawbacks. One shortcoming is that, under decentralized protocols, some agents cannot predict the group behaviour based

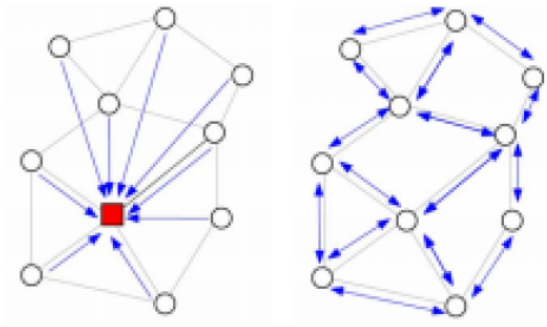


Figure 1.2: Cooperation schemes: On the left, a centralized architecture, where the center of local information fusion is denotated as a red square. On the right, a distributed architecture, where system's function is based on mechanisms of local interaction and information exhange among local nodes.

only on the available local information. Consequently, some group behaviour cannot be controlled. As a sensible example, current financial crisis actually illustrates some disadvantages of behavioural decentralization. One interesting question, therefore, is how to balance decentralization and centralization so as to further improve the overall systems performance.

1.1.3 Applications

The study of distributed control of multiple mobile robots was perhaps first motivated by the work in distributed computing , management science , and statistical physics . In the control systems society, in some pioneering works an asynchronous agreement problem was studied for distributed decision-making problems. Thereafter, some consensus algorithms were studied under various information-flow constraints.

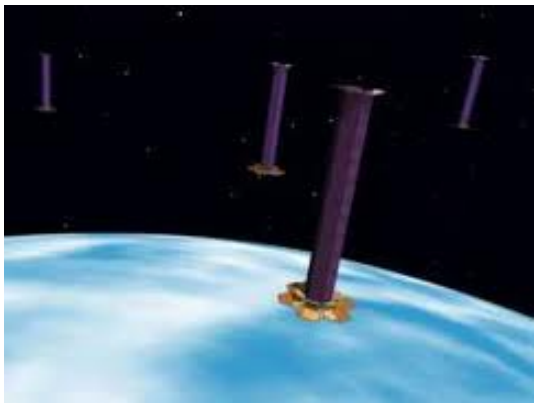


Figure 1.3: Cluster of satellites (Flight Program SPHERES,MIT)

Distributed multi-agent coordination finds application in the following categories of tasks (which are not independent, but to some extent overlapping):

1. **Consensus and the like (synchronization, rendezvous).** Consensus refers to the group behaviour that all the agents asymptotically reach a certain common agreement on a state through a local distributed protocol, with or without predefined common speed and orientation.

2. **Distributed formation and the like (flocking, swarming, containment).** Distributed formation refers to the group behaviour that all the agents form a predesigned geometrical configuration through local interactions with or without a common reference. Often agents are made to exhibit behaviors observed in nature, such as flocking birds, schooling fish, or swarming social insects

3. **Coverage.** Coverage refers to control of system's topology, in order to produce maximally spread networks without making them disconnected or exhibit holes in their coverage

4. **Distributed optimization.** This refers to algorithmic developments for the analysis and optimization of large-scale distributed systems.

5. **Distributed task assignment.** This refers to the implementation of a task-assignment algorithm in a distributed fashion based on local information.

6. **Distributed estimation and control.** This refers to collaborative distributed control design based on local estimation about the needed global information.

1.1.4 The Necessity for Global Connectivity

Mobile robot networks have recently emerged as an inexpensive and robust way of addressing a wide variety of tasks ranging from exploration, surveillance, and reconnaissance, to cooperative construction and manipulation. The success of these stories relies on efficient information exchange and coordination between the members of the team. In both consensus and formation control problems, it is often assumed that the network topology satisfies certain fundamental conditions, for example, is connected or has a directed spanning tree. However, a practical communication model is typically distance-based, i.e., two agents can communicate with each other if and only if their distance is smaller than a certain threshold, called communication range. This is particularly true for sensor networks. In order to guarantee consensus or formation control be achieved asymptotically, a connectivity maintenance mechanism is essential, which has been studied recently.

The main approach to maintaining the connectivity of a team of agents is to define some artificial potentials (between any pair of agents) in a proper way such that if two agents are neighbours initially then they will always communicate with each other thereafter. In general, the artificial potential between a pair of agents grows to be sufficiently large (could be unbounded) when the distance between them increases to be equal to the communication range. For properly designed control algorithms, which are usually

composed of the gradients of the artificial potentials, the total artificial potential is non-increasing. This then indicates that the initial communication patterns can be preserved because otherwise the total potential will become larger than the initial total artificial potential, as soon as some communication pattern is broken. Although this approach provides a systematic way to guarantee the connectivity, the corresponding control algorithms might require infinite large control inputs, which is not practical. Meanwhile, it is not even necessary to always maintain the initial communication patterns in order to guarantee the connectivity. Therefore, how to find a more effective way to guarantee connectivity deserves further investigation. An interesting problem appears when the number of initially existing communication patterns plays a role in the connectivity maintenance for the consensus problem with single-integrator kinematics (1) and control input (2). Roughly speaking, if the initial graph is "sufficiently" connected in the sense that each agent has at least a certain number of neighbours, consensus can be guaranteed to be achieved. Note that the result can only be applied to systems with single-integrator kinematics therefore further investigation is expected for systems with high-order linear dynamics or nonlinear dynamics.

In terms of connectivity maintenance for consensus and formation control, research has been devoted to continuous-time systems but practical systems are more suitable to be modelled in a discrete-time setting, which makes the study of connectivity maintenance for discrete-time systems more meaningful. In general, the connectivity maintenance for discrete-time systems is more challenging due to the fundamental limitation of the corresponding control input, which is usually piecewise constant rather than continuous. Since the problem examined in this thesis is directly related to connectivity maintenance, we will refer to this topic to a greater extend later.

Although the existing theoretical research and experiments have solved a number of technical problems in distributed multi-agent coordination, there are still many interesting, important and yet challenging research problems deserving further investigation. One of them is increasing the complexity of the cooperative task to be carried out by the group of mobile robots.

1.1.5 Mathematical Abstraction of a Networked System

Networked multi-agent systems are widely modelled as graphs where the agents are represented as nodes and edges exist between the agents that interact directly. Nodes correspond to agent and are identified unambiguously via an ordering $I = \{1, 2, ..., N\}$. Agents have their ability to move in the workspace according to their kinematic model. Location of agent i at time t is denotated as $q_i(t)$.

Communication among agents is achieved via on-board sensors with a finite range. Generally, in the following we will consider homogeneous robotic teams with identical communication range of radius R. Communication status of the network is described by the *topological* (or communication or connectivity) *graph* $\mathcal{G} = (V, E)$ over time. Edges of

the topological graph are defined through the simple rule: $(i,j) \in E(t) \Leftrightarrow |q_i(t) - q_j(t)|$. Control of the topology of the networked system is essentially control of this topological graph.

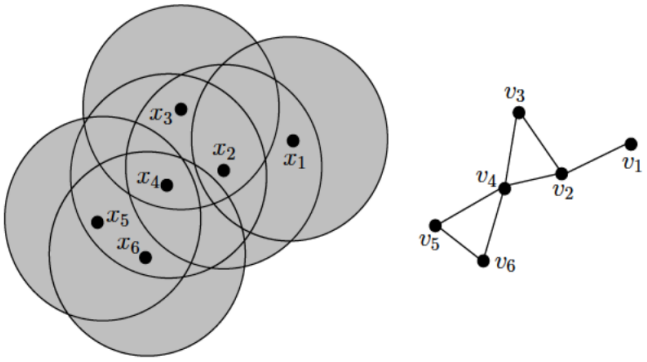


Figure 1.4: A network of agents equipped with omnidirectional range sensor viewed as a graph, with nodes corresponding to the agents and edges to the interactions

1.2 Symbolic Planning for Mobile Robots

Planning in mobile robotics aims at enabling a system to specify a motion task in a rich, high-level language and consequently convert this specification into a set of implementable low-level primitives, such as feedback controllers and communication protocols, to accomplish the desired task [21].

The initial paradigm in planning was simple specifications of the form "go from A to B while avoiding obstacles". However, this class of tasks is not expressive enough to include more sophisticated and complex motion tasks that are often of interest in modern applications. Consider, for example, applications in surveillance (e.g. "Visit A and then B infinetely often"), visiting targets sequentially (e.g. "Visit A, then B, and then C"), conditional tasks (e.g. "If you have reached A, then reach B"), or synchronisation (e.g. "Robot 1 and robot 2 should enter at the same time regions A and B respectively"). These concepts have led control scientists to pose the question: can we automatically generate provably correct control and communication strategies from task specifications given in rich and human-like language over a workspace?

Recent publications ([22],[12],[5]) have given positive results to this end both in centralized and distributed framework. "Rich" and "human-like" specifications translate naturally to formulas of expressive enough symbolic languages, like Linear Temporal Logic (LTL) and Computation Tree Logic (CTL). Originally these logics and model checking algorithms were used to specify and check the correctness of computer programs, which can be viewed as continuously operating, reactive (concurrent) systems.

Symbolic approaches of planning fit in general to a *three-level hierarchy*. To gain some intuition, let's examine this hierarchy in the case of the simplest motion task for a single robot in a workspace S:"Go from A to B while avoiding obstacles". At the first level,

the obstacle-free configuration space of the robot is partitioned into cells, and adjacency relations between cells are determined. The result is presented as a graph. Any path on this graph that starts from A and terminates at B satisfies S; hence, this is called the *specification level*. In the second place, among the (possibly infinite) paths satisfying the specification, paths are pruned accordingly to robot's mechanical and communication constraints; one robot-compatible path is chosen, according to some optimality criteria (minimal energy cost, minimal travelled distance, maximal distance from obstacles etc.). This is called the *execution level*. Both *specification* and *execution level* belong to the discrete part of the implementation. The third step, which is called the *implementation level*, consists of constructing the continuous controllers that steer the robot on the trajectory defined from the previous level. The last part is obviously continuous and is inherently related to the dynamics of the robot.

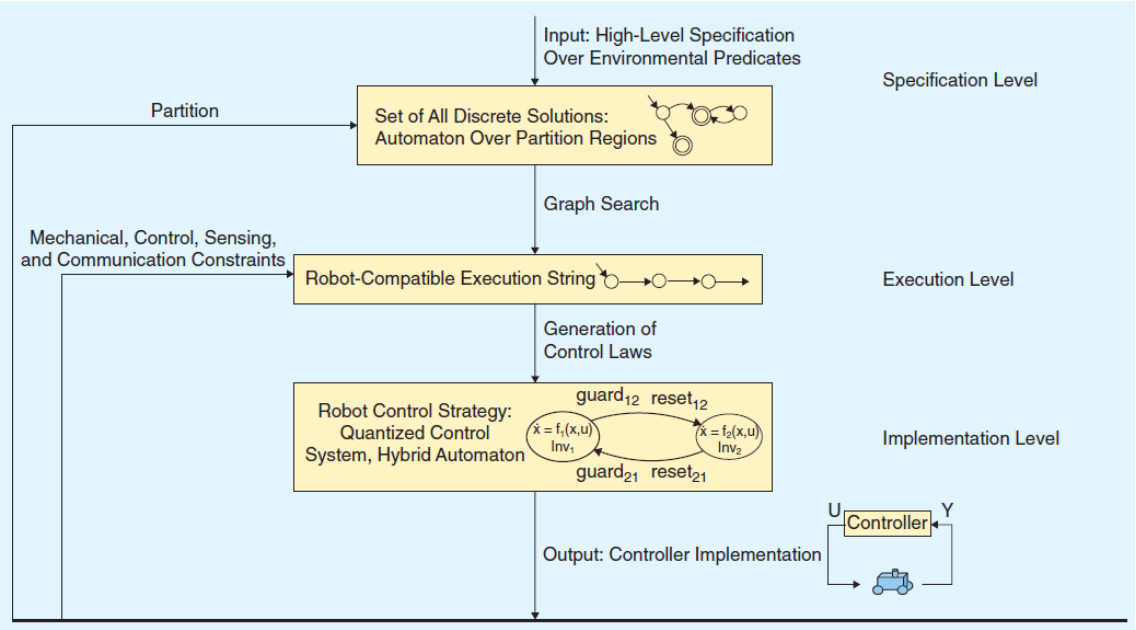


Figure 1.5: Hierarchical abstraction and computation architecture. A high level specification, such as a temporal logic formula over environmental predicates, together with a discrete graph representation of the environment, produces the set of all possible discrete solutions to the problem. A discrete execution is selected by taking into account the robot constraints, and then implemented as a hybrid automaton giving the control strategy for each robot.

The integrated scheme, namely the combination of the discrete and continuous part of planning, adds up to a *hybrid system*.

We should note here that in most approaches of symbolic robotic planning a common assumption is a discretization of the workspace. Therefore, the control of robotic motion above implementation level is reduced to the generation of a correct sequence of regions in the environment. Workspace discretization is task-dependent, since it captures constraints

regarding robots motion and sensing capabilities, obstacles etc.

In case of specifications involving logical and temporal statements, like those expressed in LTL, the specification level of symbolic planning instead of a graph results to an automaton. Hence, search performed in the execution level is more complicated than path finding and is closer to model checking. The architecture of three-level hierarchy is displayed in figure 1.5.

Extending the framework of symbolic planning to teams of robots implies computational overhead. Control of team behaviour via a common specification formula is performed through model checking in the product of all agents individual automata. Therefore most of the computations are done off-line before deployment [20]. In recent scientific works the focus is placed on minimizing the needs for communication and maximizing decentralization [7], [13].

1.3 Distributed Connectivity Maintenance

In order to accomplish almost any cooperative tasks, multi-robot systems are required to communicate among each other. Thus, preserving the connectivity of the communication graph is a crucial issue. Connectivity maintenance has been extensively studied in the last few years, usually considering kinematic agents.

Generally speaking, a method that preserves connectivity given an initially connected multi-robot network, is simply to establish that no change in local connectivity status of agents can take place. Therefore, a number of publications on the topic aims directly at maintaining *local connectivity*. In [2] authors propose a connectivity preserving distributed control law for a network of unicycles; network maintains its initial connectivity status by enforcing agents not to lose connectivity by its neighbors. However, this approach seems to produce a rather conservative solution that cannot be applied in every workspace -since in many cases it is observed that agents should reconfigure their topology in order to proceed in a workspace occupied densly by obstacles.

Hence, it is preferable to seek methodologies that offer more *robustness*. In [9], emphasis is given to networks of heterogeneous robots equipped with different sensors with limited field of view - a decentralized approach is presented for estimating an approximated minimum strongly connected digraph. In parallel, research is carried out in the field of degree regularization in multi-robot systems [29]. The issue of robustness in networked systems subject to structural changes or noise corruption and global connectivity restorage after agent removals has led to a series of interesting publications [1], [3].

In [27] authors utilize generalized energy functions and develop a control algorithm which solves the global connectivity problem in a decentralized manner via distributed connectivity estimation, without requiring maintenance of the local connectivity between robotic systems. A distributed implementation of an optimization-based method for connectivity control (via algebraic connectivity maximization) is described in [8]. Related to the last one is [32], where authors introduced a gradient-based control strategy on a class

of potential fields to guarantee that the second-smallest eigenvalue of the Laplacian matrix is greater than zero.

A very interesting approach of the problem, presented in [33] (as well as in [31]), has its roots in the area of *hybrid systems* - namely systems comprised from a continuous and a discrete component. Concretely, the idea was to develop a discrete controller (a sequence of *discrete switches*) that supervises edges addition/deletion, based on local estimates of network topology obtained by agents in a distributed way. This control scheme allows network topology control to be performed in the discrete space of graphs. When multiple deletions of links can violate connectivity tie breaking is achieved by gossip algorithms and market-based control. An experimental validation of Zavlanos-Pappas method is provided in [24].

1.4 Connectivity maintenance and Symbolic Planning

Our initial motivation for writing this thesis was to contribute in the development of a global connectivity maintenance algorithm compatible with the framework of symbolic planning. Since symbolic planning is commonly built upon discretized workspaces, a formulation of the problem in the discrete space has inherent advantages.

It seems trivial to express local connectivity relationships among agents as LTL formulas. For instance , a specification of the form S :"a_1 stay connected to a_2 forever" is expressed as follows: "GS", with G is the language symbol for "globally" and S : $|X(a_1) - X(a_2)| < R$ ($X : \mathbb{R}^n \longrightarrow \mathbb{N}$) is the location function in the workspace and R is the sensing range of the agents. Based on the previous translation local connectivity specifications can be combined into formulas and evaluated for the whole system.

However, demanding the preservation of the set of initial links among agents throughout task execution can often be proved too conservative. We need a method that allows relaxation of neighborhood relationships (or equivalently reconfiguration of the initial connectivity formulas) without violation of global connectivity - for the latter a methodology is needed that allows supervision of local connectivity relationships that are critical for global connectivity maintenance. Another challenging aspect lies in dealing with curse of dimensionality, which is soon encountered in large multi-agent system, due to the construction of the product automaton of the team for the verification of the formulas. To that end we developed a different framework, by using an innovative formulation of the problem as a well studied problem introduced in artificial intelligence, namely Constraint Satisfaction Problem.

1.5 Structure of Thesis

The following chapter contains a verbal statement of the problem taken into consideration, presents objectives of controller design and discusses its significance. In chapter 3, we formulate the problem in technical terms and present the mathematical details of

the components of our methodology. In chapter 4, we present the whole scheme of our methodology and incorporate the part of continuous controller. The 5th chapter is devoted to the demonstration and evaluation of the yielded results in two simulations, while the last chapter contains a brief presentation of potential future research directions.

Chapter 2

Verbal Statement of the Problem

2.1 Overview

Heterogeneous multi-robot systems hold promise for achieving *robustness* by leveraging on the complementary capabilities of different agents and *efficiency* by allowing sub-tasks to be completed by the most suitable agent. A key challenge is that agent composition in current multi-robot systems needs to be constant and pre-defined. Recent approaches are motivated by the need for increased robustness, heterogeneity and reconfigurability in future multi-robot setups.

Targeted attributes of modern approaches are:

- **Increased robustness**. The set of heterogeneous agents with different capabilities is robust against faults of the individual agents (as any multi-agent system) but also against design failures in an individual type of agent, for example, if the kinematics of a certain agent does not allow a particular task, or a sensor system of a particular type of an agent fails because of environmental conditions - for example, if the intensity camera of an agent fails because of poor illumination, the depth camera of another could still work. Hence, this attribute is quantified by the multitude and the magnitude of the faults/uncertainties the system is able to handle.

- **Increased efficiency**. The non-uniformity of agents allows inexpensive agents to be used for tasks achievable by simple agents (e.g. coverage of the visual scene) so that agents with expensive capabilities, such as manipulation, can be used more efficiently. This attribute is quantified in terms of the time required to accomplish a task.

- **Online adaptation** of the individual agent objectives and controls based on linguistic information exchange and implicit sensor based coordination. Current multi-robot logic based methods do not address reconfigurability when the verification fails. In our formulation, we consider adaptation of the individual controller and task to be accomplished both in the continuous and the discrete level. This attribute is

quantified in terms of the load of change required at the individual controller or task updates.

- **Lean communication** requirements. Because communication takes place at the symbolic level, there is no need for continuous exchange of large data and over-networking becomes unnecessary for the systems in hand. An important contribution of the proposed approach is thus that lean communication requirements are adequate for the correctness of the overall framework. This attribute is thus quantified in terms of the required bandwidth to exchange data online between the agents.

Reconfigurable navigation is based on the development of distributed navigation schemes for the complex multi-agent system setup that incorporate in the feedback loop the updated knowledge of the environment that has been accrued through learning and implicit sensing information, based on their cognitive capabilities, and this provides for a direct integration of cognitive information in the task planning design of multi-agent systems.

2.1.1 Multi-agent Navigation

i. State of the art

Cooperative control of multiple robots has received considerable attention in the last decade owing to a wide range of applications such as moving a large number of objects, environmental monitoring, rescue missions, distributed transportation, and multi-point surveillance; tasks that cannot be efficiently accomplished by a single robot. In such cases, the robots are spatially distributed and work together based either on commands given by a supervisor in a centralized scheme or following some rules and communication strategies designed in advance in a distributed scheme. Broadly speaking, the latter falls within the domain of decentralized control. Considering the communication limits, calculation costs, and required devices to provide a perfect knowledge for a centralized planner, there are many real world problems in which a decentralized controller can be found preferable. Moreover, simple, local motion coordination rules at the individual level resulting in remarkable and complex intelligent behavior at the group level are desirable for large scale robotic systems.

When approached from a control point of view, particularly in situations where the agents have competing interests or seek different goals, the coordination problem becomes difficult to deal with. First, it is important to control robots with different hardware and software so as to add robustness to the formation. Moreover, taking dynamic environments and uncertainty external to the multi-robot system itself into account, heterogeneous systems, with robots in different shapes and abilities, are more applicable in real world applications than systems with the same team members. In addition, the control methods need to be computationally inexpensive, taking the real-time response to the environment into consideration. Furthermore, in respect of autonomous decentralized control, how to control robots in the most suitable formation considering the ability of each robot is an-

other problem. Besides, communication is limited in dynamic environments, especially in the outdoor case.

The concept of coordination of multiple robots has been extensively studied in the recent. Some of the existing classical motion planning methods include potential function approach and optimization-based approach. Moreover, the idea of artificial potential functions for obstacle avoidance, which was proposed in [25], was extended in [11],[10] for multiple agents.

ii. Beyond the state of the art

Despite the recent progress in multi-agent coordination and navigation, certain issues concerning lack of global knowledge about the environment and the goal objective, as well as limited resources and coordination constraints between the agents, still remain open. In this respect, updated knowledge of the environment, accrued through either learning from explicit communication with other agents or sensing, and implicit sensing information call for the need of integration, in order to modify the individual agent controllers in an on-line manner to fulfill the task in a desired manner.

For example, updating individual knowledge about the environment via explicit communication, detecting the motion intention implicitly by simply watching motion trends or interpreting force/torque measurements in a manipulation task as motion intentions and adapting the individual agent controller accordingly, are major steps towards healing the aforementioned deficiencies in the current literature. Agents replanning due to updated knowledge of the environment [18], or motion revision due to infeasibility of specifications [17] are issues that recently have been taken into investigation within the scientific community. To our knowledge, albeit progress is noted in the field of revising motion in the workspace due to infeasibility, no results have been presented yet regarding *reconfiguration directly in the space of systems specifications* (or re-engineering specifications).

Such integration requires a drastically different new approach to traditional distributed control approaches, which will actually be our main focus in this thesis. Our goal will be to design an autonomous system consisting of multiple agents aiming at solving tasks using interactions among them both explicitly and implicitly. To that end, specification satisfaction will lead us to formalize the problem in the discrete level as a Constraint Satisfaction Problem (CSP) - associating a problem from artificial intelligence with discrete controller synthesis. Towards the implementation of the continuous part of the scheme, notions from Prescribed Performance Control are used, so as to guarantee the maintenance of topological properties controlled in the discrete level, as well as satisfaction of the goal with the desired manner obeying simultaneously operational constraints. Finally, analysis of connectivity with elements from graph theory will be integrated.

2.1.2 Coordination in Unknown Workspace

In a multi-agent coordination task, for instance, one challenging scenario is to deploy a team of (possibly heterogeneous) robots in an unknown workspace and assign to them an

exploration or surveillance mission. A plausible solution to the problem could be to assign to the most suitable agent (the *leader*) the task of producing a path to be followed in the workspace and guide the rest of the networks (the *followers*) according to that. A major issue is that the followers maintain connectivity with the leader, in order not to break off from the network and to be able to contribute to collaborative tasks throughout motion.

Additionally, let's assume that we initially have a predefined set of neighbours for each of the agents - or equivalently an initial (globally connected) network topology-, namely that each agent has to stay in contact with a specific subset of the network during motion. How such a scheme would react in case it faces obstacles in the workspace in real-time? It is obvious that the system will get stuck due to its rigid topology when the free space in the neighborhood of obstacles is insufficient (e.g. imagine a triangular formation trying to enter a room from a narrow corridor). Then, the leader has to backtrack and search for another appropriate path in the workspace. This implies further computational effort for the leader and increases the time of task accomplishment, or even prohibits the execution of the motion task.

Reasonably, a more efficient approach would be to allow the network reconfigure itself in regions where, due to workspace limitations, it is impossible to preserve the initial connectivity status. Thus, the system will not have to seek for an alternative path, but keep moving on leader's current path, while seeking for an appropriate topological reconfiguration. An additional desired property would be to enable the network *restore its original topology* in the future, whenever it is allowed.

The above proposed framework comprises of two categories of specifications:

(i.) Soft **local connectivity specifications**. These specifications could be relaxed by need, under the restriction that *they do not violate global connectivity.*

(ii.) Hard **collision avoidance specifications**. These specifications cannot be relaxed at no point of the task execution.

Assuming a proper discretization of the workspace, the path planned by the leader will be a sequence of partitions (abstracted by *points* that correspond to their centres) that should be visited in specific order. In that context, the reconfigurable coordination scheme will be a hybrid system composed of two components:

- A **discrete controller** supervising discrete changes in network's connectivity graph. Discrete controller is activated in each discrete step of leader and controls network's configuration during the next step.

- A **continuous controller** which deals with agents motion between two succesive points of leader's path. The continuous controller maintains the topological configuration as defined by the discrete controller and ensures collision avoidance.

2.2 Discrete Controller Objectives

Controller synthesis in the discrete domain should be enabled to tackle the following issues:

- **Controller synthesis** for each individual agent **from individual specifications** (connectivity / collision avoidance)
- **Specification adjustments due to updates** from implicit/explicit communication (e.g. detection of obstacles in agent's environment)
- **Gradual verification at meeting events** with other agents in the group and consensus to the next topological status
- **Reconfiguration strategies** at the discrete level in the event that mutual satisfiability of agents plans are not possible

The discrete controller of our scheme should reason about *global* connectivity (a property for the whole system) from individual *local* specifications. Moreover, it should be able to reduce the restrictions imposed by the connectivity maintenance control to the necessary minimum, which means that it should not impose a specific minimal set of neighbors for each agent, in order to allow maximal flexibility to the system - the notion of *minimal connectivity* used in our work deviates from definition found in [28]; simply put, we consider a globally connected network graph that is able to drop local connectivity edges by need as far as they do not violate global connectivity. Actually, if necessary, discrete controller can drop the whole set of neighbors of a specific agent (*transient loss of connectivity*), if it can assign to the agent different neighbors that allow to the system to proceed whithout violation of global connectivity.

2.3 Motivating Example

We briefly present a motivating example of multi-robot coordination subject to specifications reconfiguration, omitting mathematical details.

As shown in figure 2.1a, leader follows a given plan (sequence of points) in the workspace until it reaches the target or system gets stuck. Followers try to maintain their given local connectivity specifications. Initially, we have:

- ATOMIC SPECIFICATIONS
  - $a0$ **(Leader)**: Go to target T, No collision
  - $a1$ **(Follower)**: Stay connected to $a0$, No collision
  - $a2$ **(Follower)**: Stay connected to $a0$, No collision
- SYSTEM SPECIFICATIONS
  Maintain global connectivity

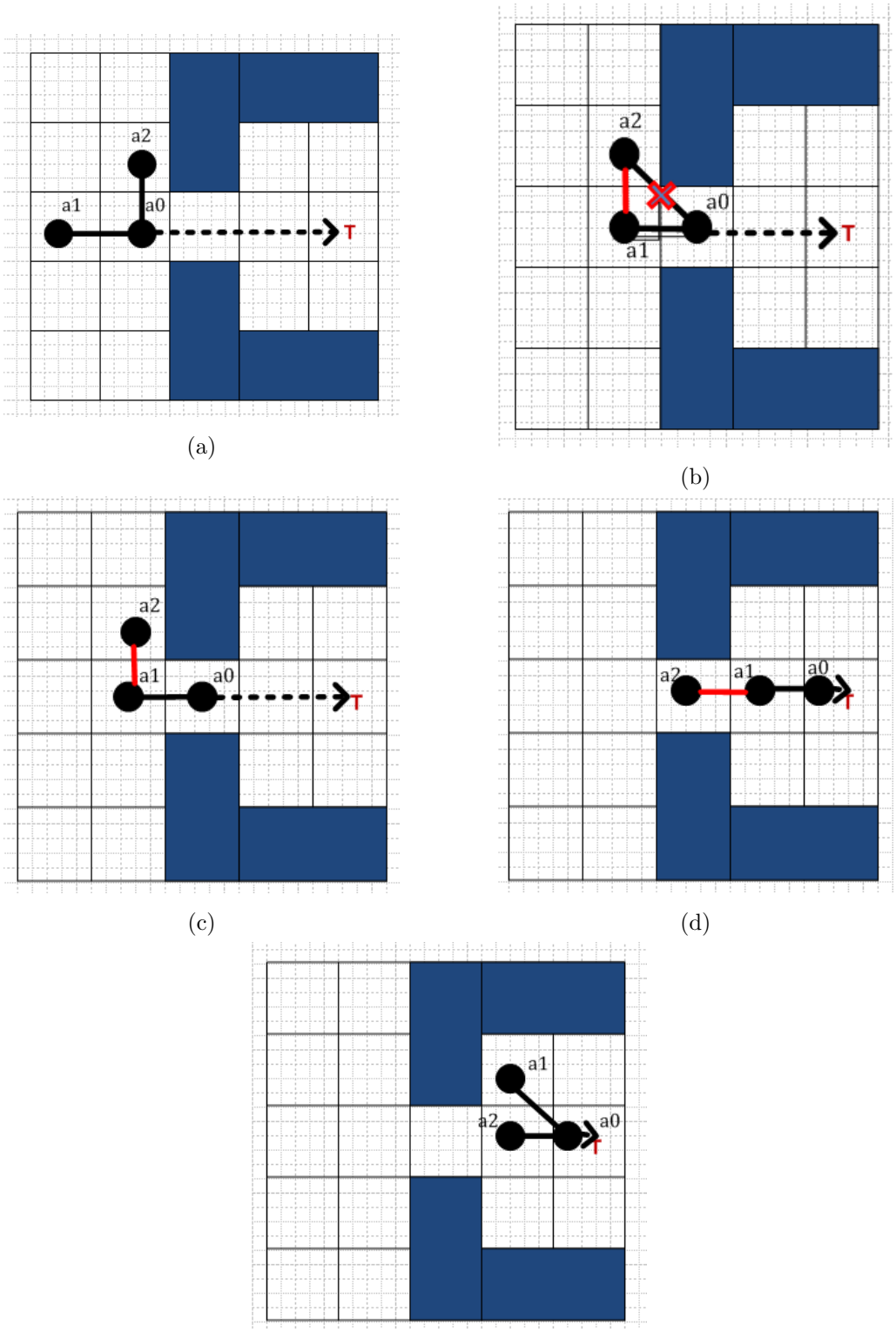


Figure 2.1: Successive plots of a motion task, where specifications reconfiguration is needed. Regions occupied by obstacles are painted in blue. Initial connectivity specifications are depicted in black, while edges added after reconfiguration are depicted in red. Dotted line represents leader's predefined path to the target

System specification is implicitely controlled via atomic specifications - this is to say that before removal of an individual connectivity specification, the specification is checked in order to find out whether it violates the requirement for global connectivity.

The set of local connectivity specifications is abstracted as a graph and corresponds to the required connectivity graph of the network - which does not neccessarily coincides with the topological (or communication) graph of the network. Actually, the required connectivity graph should always be a subgraph of the topological graph, in order the set of specifications not to be violated. Namely, with the term *topological* graph we refer to a graph with edges between agents that can communicate, while with the term *specifications* graph we refer to a graph with edges between agents connected through specifications. For instance, in figure 2.1a, if we assume that the communication range for each agent is its 8-squares neighborgood, edge $a1$ - $a2$ does belong to the topological graph, albeit it does not belong to the specifications graph.

Note that each agent has the ability to communicate if needed with the agents inside its sensor range, namely the set of neighbors in its connectivity graph (although not restricted to do) - which includes the set of agents due to constraints.

After the first step, in figure 2.1b, the multi-robot team should enter a room via a narrow door. Agent $a2$ detects the obstacle and decides that it cannot move while staying connected with $a0$ - since moving one square downwards is conflicting with collision avoidance specification with $a1$. As a result, the system should reason about conflicting specifications, in particular collision avoidance between $a2$ and $a1$ and local connectivity between $a2$ and $a0$ and reconfigure specifications set. Since collision avoidance specifications are hard, local connectivity should be violated. However, removing edge $a0$ - $a1$ causes violation of global connectivity in the new specifications graph. As a result, the system looks for adding a new edge from the available topological graph edges to cure this removal. Hopefully, it adds a new local connectivity specification between $a2$ and $a1$.

The new specifications set will be :

- ATOMIC SPECIFICATIONS
  - $a0$ **(Leader)**: Go to target T, No collision
  - $a1$ **(Follower)**: Stay connected to $a0$, No collision
  - $a2$ **(Follower)**: Stay connected to $a1$, No collision
- SYSTEM SPECIFICATIONS
  Maintain global connectivity

The formation of robots manages to proceed to the target under the reconfigured topology (2.1c, 2.1d). Upon leader's arrival at the final destination, in case of enough free workspace, agents will look for a solution closer to their initial specifications set and restore initial connectivity status, as in 2.1e .

Chapter 3

Technical Problem Statement

3.1 Discrete Controller

Our scheme's **discrete controller** is called in discrete instances during system's motion and controls the topology of the network. Thus, it is a step-by-step algorithm with:

INPUT

- Leader's next position on path
- Sensing information (real-time information gathering from on-board sensors : obstacle detection, agents positions)
- Agents local connectivity and collision avoidance specifications

OUTPUT

- Next configuration of the system (preserving global connectivity)

OBJECTIVES

- Specifications maintenance (or *reconfiguration* when system gets stuck, i.e. relaxation of specifications without global connectivity violation)

3.2 Reduction to CSP

By inspection of figure 3.1 we can gain some intuition regarding the mathematical formulation of the problem. Our input is a set of **variables**, each of which associated with :

-a set of possible values (**domains**) defined by mobility constraints of each agent, and

-a number of **constraints** (local connectivity and collision avoidance) defined over the above-mentioned domains.

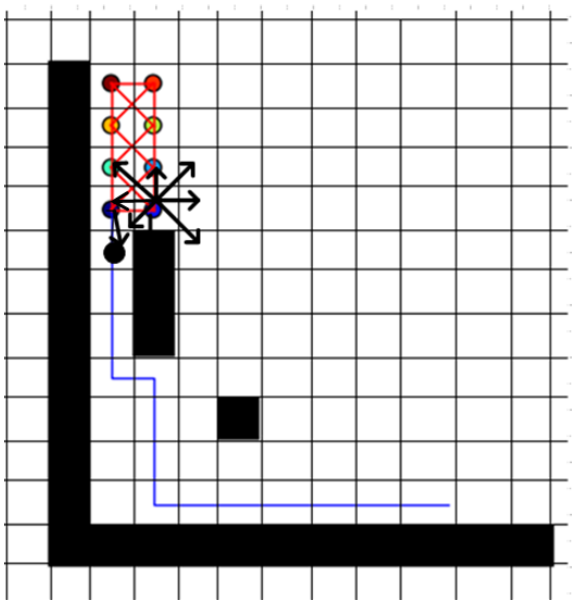


Figure 3.1: The mechanism underlying the discrete controller. Given initial positions, leader's next position (over leader's predefined path) and initial connectivity status, each of the agents has to decide among the positions it can move to according to its specifications set.

Expressing these constraints as boolean functions in a straightforward manner, we obtain a well-studied problem in the field of artificial intelligence, namely **Constraint Satisfaction Problem (CSP)** ([26]).

3.2.1 CSP

A CSP is a tuple $< X, D, C >$, consisting of

- $X = \{x_1, x_2, ..., x_n\}$: a set of n *variables*,
- $D = \{D_1(x_1), D_2(x_2), ..., D_n(x_n)\}$: a set of respective finite, discrete *domains*, and
- $C = \{C_1(x_{11}, x_{12}, ..., x_{1j_1}), C_2(x_{21}, x_{22}, ..., x_{2j_2}), ..., C_m(x_{m1}, x_{m2}, ..., x_{mj_m})\}$: a set of predicates C_i(or *constraints*) defined on the Cartesian product $D_{i1} \times D_{i2} \times ... \times D_{ij}$.

A predicate C_i is true if and only if the assignment of its values satisfies the constraint (it can be viewed as a boolean function).

Solving CSP is equivalent to finding an assignment of values to all variables such that all constraints are satisfied.

3.2.2 Coordination as CSP

We are ready now to express our coordination under individual specifications problem as CSP. We will apply this reformulation for the example of figure 3.1, although it is straightforward to generalize the reduction for different tessellations and robots.

- $X = \{x_1, x_2, ..., x_n\}$, where $x_i \in \mathbb{Z}^2$ is the current position of agent i in the workspace,
- $D = \{D_1(x_1), D_2(x_2), ..., D_n(x_n)\}$, where $D_i(x_i)$ is the set of regions where agent x_i can move during the next step, according to its mobility constraints - in our example D_i is the subset of 9-square neighbors of x_i (including current position) which is not occupied by obstacles
- C is the set of
  - Collision avoidance constraints $C_{coll.av.}(i,j) = (x_i \neq x_j), \forall i,j : i \neq j$
  - Local connectivity constraints $C_{con}(i,j) = (max(|x_{i1} - x_{j1}|, |x_{i2} - x_{j2}|) == 1)$

3.2.3 Distributed CSP

A distributed CSP (DCSP) is a CSP in which the variables and constraints are distributed among automated agents.

In this section we loosly follow presentation of formulation and proposed distributed alforithm for DCSP found in [30].

We assume the following communication model:

- Agents communicate via message passing. An agent can send a message to another agent if and only if it knows the position of the other.
- Delays in message passing are finite, though random, and messages are received in the order they are sent.

Each agent has a set of variables and tries to decide for a value among them in order the final collection of values of the whole system to satisfy problem constraints. Without loss of generality, we assume that (i.) each agent has exactly one variable, (ii.) constraints are binary and (iii.) each agent knows all constraint predicates associated with its value. Moreover, we assume that a unique identifier is assigned to each agent.

It is straightforward to develop centralized and synchronous backtracking methods that seek for a solution of a DCSP. However, the non-trivial *asynchronous backtracking* algorithm of Yokoo et al. owns a set of desirable properties, such as

- agents act asynchronously , concurrently, based on local knowledge, without global control,
- less information is communicated, succeeding security/privacy of agents, and
- it is sound and complete, namely algorithm terminates with failure if and only if the CSP has no solution (*overconstrained*)

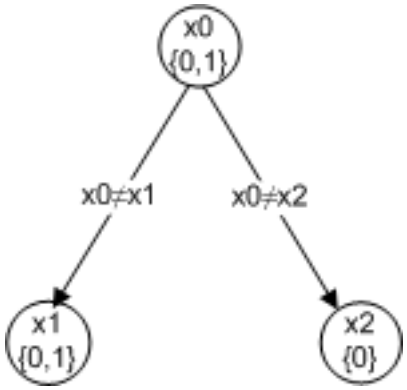


Figure 3.2: Example of constraint network

3.2.4 Asynchronous Backtracking

A DCSP can be visualized as graph G with nodes problem variables x_i and edges between nodes associated with constraints (see for instance 3.2).

Each agent instantiates its variable concurrently and sends its value to the agents which are connected by outgoing links. After that, agents wait for and respond to messages of two categories:

1. ***ok?*** **messages**, that a constraint-evaluating agent receives from a value-sending agent asking whether the value currently chosen is acceptable, and

2. ***nogood*** **messages**, that a value-sending agent receives, as an indication that the constraint-evaluating agent has encountered a constraint violation.

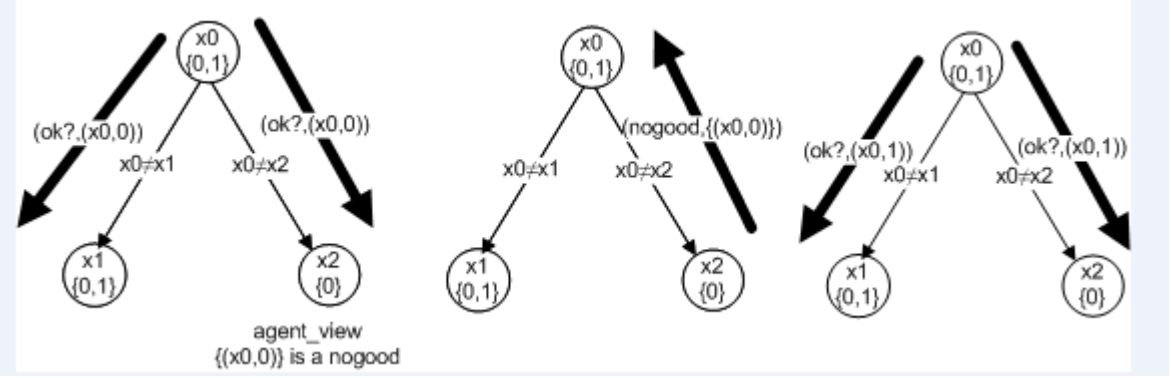


Figure 3.3: Example of asynchronous backtracking

Each agent receives a set of values from the agents that are connected by incoming links. These values constitute its `agent_view`. Whenever an *ok?* message is received, the evaluating agents adds it to `agent_view` and checks whether its own value assignment is consistent with the updated `agent_view`. Consistency happens if all constraints the agent evaluates are true under the value assignments described in `agent_view` and (x_i,

current_value), and if all communicated *nogoods* are not compatible with agent_view and (x_i, current_value).

(i). **if** *received(**ok?**,(x_j, d_j))* **then**
  add (x_j, d_j) to agent_view ;
  check_agent_view
end
(ii). **if** *received(**nogood**,x_j, nogood)* **then**
  add nogood to nogood_list;
  if *(x_k, d_k) where x_k is not connected is in nogood* **then**
    request x_k to add a link from x_k to x_i ;
    add (x_k, d_k) to agent_view
  end
  old_value← current_value;
  check_agent_view;
  if *old_value=current_value* **then**
    send(**ok?**,(x_j,current_value)) to x_j
  end
end
(iii). **function** check_agent_view ;
if *agent_view and current_value are not consistent* **then**
  if *no value in D_i is consistent with agent_view* **then**
    backtrack
  else
    select $d \in D_i$ where agent_view and d are consistent ;
    current_value ← d ;
    send (**ok?**,(x_i,d)) to outgoing links
  end
end
(iv). **function** backtrack ;
nogoods ← $\{V|V =$ inconsistent subset of agent_view $\}$;
if *an empty set is an element of nogoods* **then**
  broadcast to the other agents that there is **no solution** ;
  terminate
end
for *each $V \in$ nogoods* **do**
  select (x_j, d_j) where x_j has the lowest priority in V ;
  send (**nogood**, x_i, V) to x_j ;
  remove (x_j, d_j) from agent_view;
end
check_agent_view

Algorithm 1: Asynchronous backtracking algorithm

A subset of agent_view is a *nogood* if the agent is not able to find any consistent value

with the subset. If an agent discovers that a subset of its own `agent_view` is a *nogood*, then *the assignments of other agents should be changed.* Therefore, the agent causes a backtrack and sends a *nogood* message to one of the other agents.

For example, in figure 3.3, initially x_0 instantiates its value to 0 and sends *ok?* messages to agents x_1 and x_2. x_1 receives the message and reacts by instantiating its value to 1 independently of its previous assignment. x_0 can only assign its value to 0, which contradicts to assignment of x_0, due to the constraint $x_0 \neq x_2$. Consequenlty, x_2 sends a `nogood` message to x_0. x_0 reacts by changing its value to 1 and the solution is found.

To avoid infinite loops in the network we use a total order relationship among nodes. Concretely, we define a priority order among agents by using the alphabetical order of their unique identifiers. Links are directed according to that order (from the higher to lower priority agents). Thus, *ok?* messages are communicated from higher priority to lower priority agents and *nogoods* in the opposite direction.

Since agents act asynchronously, an `agent_view` is subject to incessant changes. This leads potentially to inconsistencies, because a constraint-evaluating agent might send a *nogood* message to an agent that has already changed the value of an offending variable due to other constraints. The phenomenon of inconsistencies is healed through the use of *context attachment* in the procedure of checking `nogood` messages- namely, firstly the recipient checks compatibility of message with its current `agent_view` and assignment. A `nogood` can be viewed as an additional constraint among agents (which are not necessarily neighbors in the constraint network). This is of course a logical binding, not a physical one.

Theorem 3.1. *If there exist a solution, this algorithm reaches a stable state where all variable values satisfy all the constraints and if no solution exists the algorithm discovers this fact and terminates.*

Soundedness and completeness of the algorithm are clear. Because a *nogood* logically represents a set of assignments that leads to a contradiction, an empty *nogood* means that any set of assignments leads to a contradiction. Thus the algorithm terminates with failure if and only if an empty *nogood* is found.

Termination of the algorithm in finite time is proved by induction, through the use of the priority order of agents.

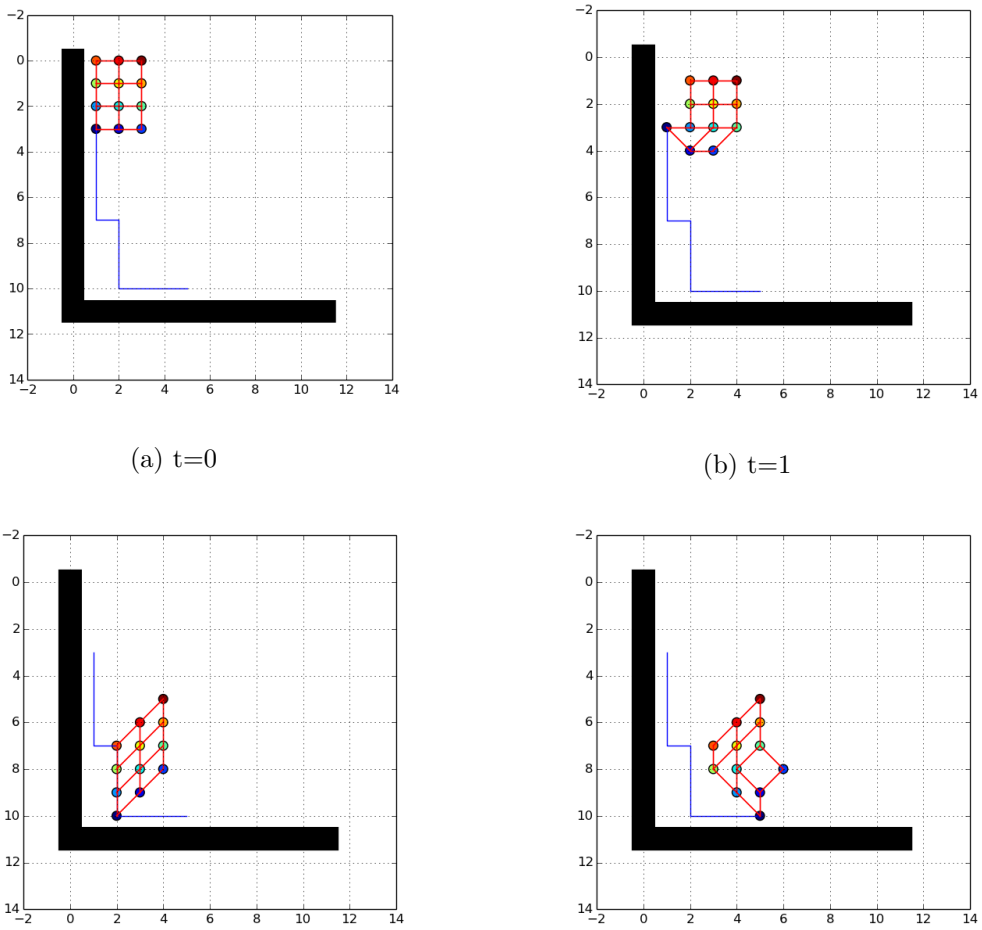
3.2.5 Weak-Commitment Search Algorithm

A more efficient distributed approach to the problem can be achieved by exploiting appropriate heuristics that allow the abandonment of a partial solution, if no consistent value with that partial solution exists. In that version, the algorithm uses the min-conflict heuristic as a value-ordering heuristic.

3.2.6 Complexity

Constraint satisfaction is NP-complete in general. Worst-case time complexity of presented distributed algorithms is exponential in the number of variables n. Worst-case space complexity is determined by the number of recorded *nogoods*. Since an agent can forget old *nogoods* after the creation of a new one and gets rid of incompatible *nogoods*, each agent x_i needs to record at most $|D_i|$ *nogoods*.

3.2.7 Example



(a) t=0

(b) t=1

(c) t=9

(d) t=12

Figure 3.4: Steps of network configurations during motion task execution

We present a single example, where the network executes a motion task in a workspace very sparsely occupied by obstacles. Therefore, need for topology reconfiguration is not encountered. Followers proceed by deciding their next step in a distributed manner, via solving repeatedly the DCSP for each step of the leader.

It is observed that the network does not maintain formation. Concretely, it seems

to change formation in a rather counterintuitive way (while preserving network topology status). This should be attributed to the random way in which each agent seeks for solution in the DCSP at each leader's step. Formation keeping can be easily achieved by introducing memory to the system. In particular, a valid solution would allow each agent to keep moving at the same direction with its previous step, while it does not senses obstacles. This technique would moreover reduce the necessity of constantly solving DCSP for identical instances of the problem (e.g. instances in a workspace which is free of obstacles).

3.3 Reconfiguration

Let's briefly summarize what we have achieved so far: We have designed a distributed discrete controller that outputs networks next configuration under a specifications set if and only if such a configuration exists. Otherwise, controller decides in finite time that a solution to our problem does not exist, because the problem is *overconstraint*.

In this section, we will propose methods to overcome these dead-ends through problem reformulation. In particular, we are going to relax some of the agents specifications until we settle at a feasible relaxed version of the original problem. Since collision avoidance cannot be relaxed, we target at local connectivity specifications. As we have already mentioned, global connectivity maintenance is vital for the multi-robot system. Consequently, we are going to relax local connectivity specifications, while checking that they do not imply violation of global connectivity. Thus, we need a structured method to characterise edges according to their effect on global connectivity. In the following, we present a distributed procedure to do it.

3.3.1 Distributed Estimation of Adjecency Matrix

The main idea in this stage is that each agent obtains in a decntralized way an estimate of the graph adjacency which they update by communicating with its neighbors. During this update process it is assumed that the graph topology doesn't change. Although this assumption is rather controversial, the assumption will work fairly well as long as the graph remains connected during this process, even if its topology changes.

Thus, if A^k is the estimate of the adjacency matrix made by agent k, we will refer to the $(i,j)^{th}$ element of the matrix at the p^{th} step of updating it by ${}^pA^k_{i,j}$. There are two processes now:

a) *Initiation:* ${}^0A^k_{i,j}$ is initiated for each agent k by the following rule:

$${}^0A^k_{i,j} = \begin{cases} 1 & \text{, if either } i \text{ or } j \text{ is connected to agent } k \\ 0 & \text{, otherwise} \end{cases}$$

b) *Propagation:* The elements of adjacency matrix are updated by talking to the neigh-

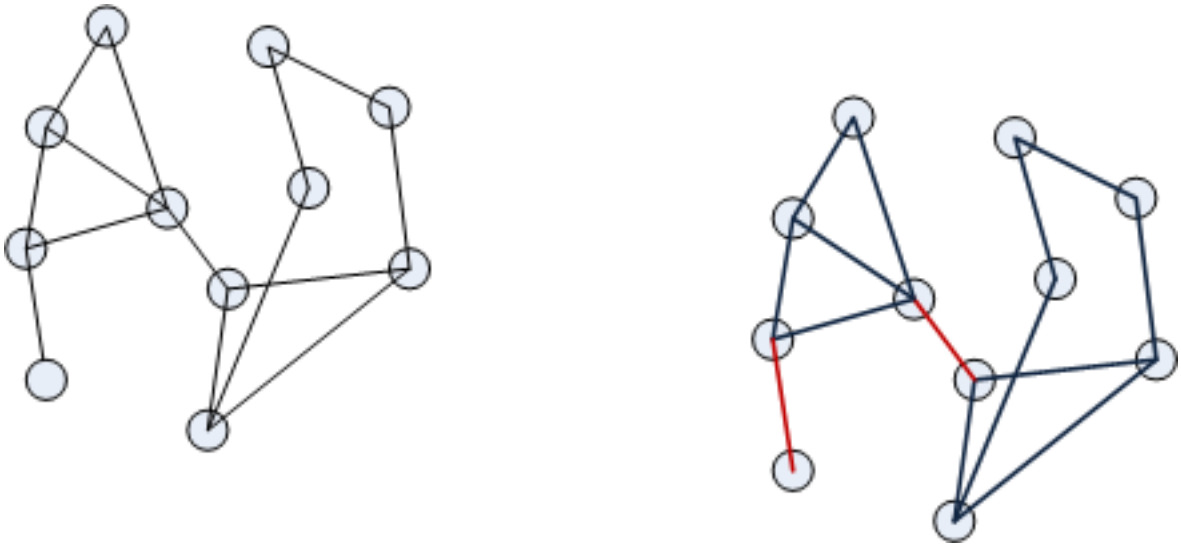


Figure 3.5: Network graph. On the right, critical edges of the graph are depicted in red, while noncritical edges are depicted in blue.

bors according to the following rule:

$$^{p+1}A^k_{i,j} = \max\{^pA^k_{i,j},\ \max_{l\in N_k}\ ^pA^l_{i,j}\}$$

where, N_k is the set of neighbors of agent k.

It can be proved ([1]) that the number of steps that the process of propagation will take to be completed (i.e. for an agent to obtain the whole adjacency matrix of the connected component of the graph) is not more than the length of the longest path in the particular connected component of the graph. Since the agents don't have an estimate of the length of the longest path, we run the propagation for N times, which can be the maximum length of any path in a grap with N nodes.

Moreover it is to be noted that the information about the adjacency matrix that gets propagated is the one that was initiated at the *initiation* step. Thus even if the graph topology gets changed during the *propagation* process, if the graph remains connected along this time, the final estimate that each agent will have about the graph adjacency is the one that was existent at the time of *initiation* step.

3.3.2 Finding the *critical edges* connected to a particular agent

Once each agent has an idea about the adjacency of the particular connected component of the graph, they should determine which ones are the *critical edges* that are connected to itself. So here we define a *critical edge* of a connected component of a graph to be *such an edge, the removal of which will break the particular connected component into smaller disconnected components.* This notion is illustrated in figure 3.5.

With the above definition for a *critical edge*, we can tell if an edge is critical or not in two trivial ways:

i. Remove the particular edge in question and compute the Laplacian of the newly formed graph. Compute the second eigenvalue of this Laplacian. If this eigenvalue is zero then the edge is a critical edge.

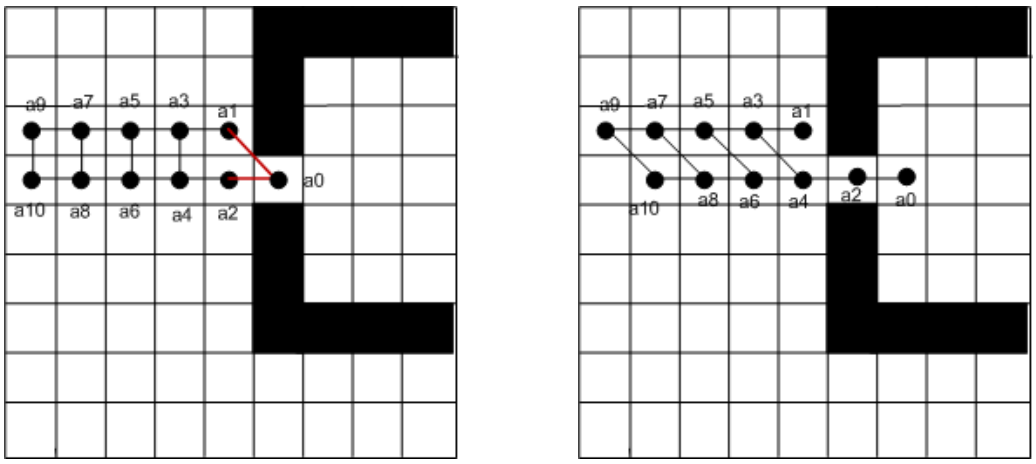
ii. Remove the particular edge in question and compute the adjacency matrix $\tilde{A}$ of the newly formed graph. Compute $\tilde{A}^s = \tilde{A} \cdot \tilde{A} \cdots \tilde{A}$ (s times), for all $s \in \{1, 2, \ldots, N\}$. If at least one of these powers of $\tilde{A}$ is a positive matrix (i.e. a matrix with all the elements being positive), then the particular edge is not a critical edge, else it is.

In practice the second method is computationally less expensive.

We define $C_i \subseteq N_i$ the subset of neighbors to agent i, the connections to which form critical edges for the particular component of the graph. Agent i will be responsible in ensuring maintenance of only the critical edge that connect i and $j \in C_i$.

3.3.3 Minimal Unsatisfiable Cores

For an efficient relaxation of the problem, in the first place we need a *reasoning of infeasibility in the level of specifications*. To that end we will need the notion of Minimal Unsatisfiable Cores(MUCs) (introduced in [19],[15]).



(a) t=0 (b) t=1

Figure 3.6: Task subject to specifications reconfiguration. At t=0, MUC is extracted - comprised from indicated local connectivity and respective collision avoidance specification. Discrete controller selects to reconfigure specifications by dropping (noncritical) local connectivity of a_0 and a_1

Definition 3.1. *Let* $P =< X, D, C >$, $P' =< X', D', C' >$ *be two CSPs.* P' *is an* unsatisfiable core *of* P *iff* P' *is unsatisfiable,* $X' \subset X \wedge D' \subset D \wedge C' \subset C$.

Different unsatisfiable cores of a given CSP may exist. Those which do not contain any proper unsatisfiable core are called minimal.

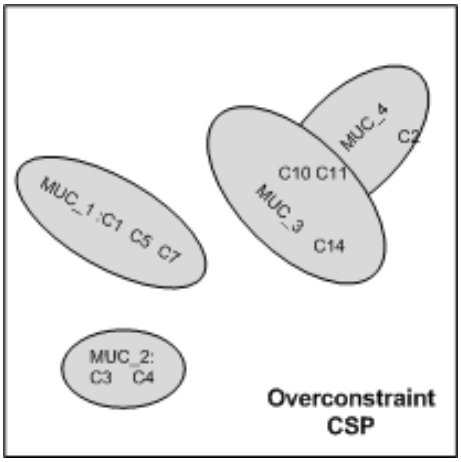


Figure 3.7: Overconstraint CSP with MUCs shown. Intersections between MUCs may be non empty, meaning that MUCs may share common constraints.

Definition 3.2. *Let $P =< X, D, C >$ be a CSP and $P' =< X', D', C' >$ be an unsatisfiable core of P. P' is a* Minimal Unsatisfiable Core (MUC) *of P iff it does not exist any unsatisfiable core P'' of P' s.t. $P'' \neq P'$.*

Every infeasible CSP contains at least one MUC. In order to reach a feasible relaxed version of a CSP, we should iteratively get rid of existing MUCs. Thus, extracting MUCs is a valuable procedure in reengineering a CSP, since dropping one constraint within the MUC reduce the number of MUCs in the obtained relaxed problem by at least one.

3.3.4 Extracting MUCs

In order to extract a minimal core is necessary to iteratively identify the constraints that are involved in it. Concretely, we know that given an unsatisfiable CSP P and an (arbitrarily defined) total ordering of the constraints $C_1, C_2, ..., C_m$, there exists a constraint C_i such that $P_{\downarrow\{C_1,...,C_{i-1}\}}$ is satisfiable and $P_{\downarrow\{C_1,...,C_i\}}$ is unsatisfiable. Then C_i belongs to a MUC, is called a *transition* constraint of P, while a MUC is contained in $P_{\downarrow\{C_1,...,C_i\}}$ - as a result, constraint C_j with $j > i$ can be safely removed.

In comparison to constructive and destructive approaches in order to identify a transition constraint, a more efficient approach is the dichotomic search:

dcTransitionConstraint(P:CSP,k:int):Constraint
min← k+1;
max← $|C|$;
while $min \neq max$ **do**
  center ← (min+max)/2;
  if $P_{\downarrow\{C_1,...C_{center}\}}$ *is satisfiable* **then**
    $min \leftarrow$ *center+1;*
  else
    $max \leftarrow$ *center+1;*
  end
  return C_{min} **end**

Algorithm 2: Transition Constraint Extraction

To get a second element of the MUC, we apply the same function on a new CSP P' which is obtained from P via removing all C_j such that $j > i$ and defining a new order where C_i is considered as the first element. This iterative process is terminated when all constraints of the MUC are discovered:

dcMUC(P:CSP):CSP
$P' \leftarrow$ P ; k $\leftarrow 0$;
while $k < |C'| - 1$;
do
  $C_i \leftarrow dcTransition(P', k)$;
  k $\leftarrow k + 1$;
  $P' \leftarrow P^{\uparrow\{C_j|j>i\}}$;
  tmp← C_i;
  $C_{j+1} \leftarrow C_j$ for $1 \leq j < i$;
  $C_1 \leftarrow tmp$;
  If $P'^{\uparrow C_{|C'|}}$ is unsatisfiable **then**
  return $P'^{\uparrow C_{|C'|}}$
end

Algorithm 3: MUC extraction

The worst case number of calls to the CSP solver is $O(log(e)k_e)$, where $e = |C|$ and k_e the number of constraints of the extracted MUC.

3.3.5 Problem Relaxation

Problem relaxation is performed iteratively while CSP is unsatisfiable. MUCs are iteratively extracted and relaxed through dropping soft constraints. Whenever a MUC consisting only of hard constraints is extracted, the system cannot proceed further (since either global connectivity or collision avoidance should occur). In such a case leader has to wait in its present position (so as the system becomes more connected) or search for

another path. The latter is not expected to take place in connected workspaces.

Relax(P:CSP):CSP
while *P= (X,D,C) is overconstraint;*
do
  MUC $\leftarrow$ $dcMUC(P)$;
  if $c \in MUC : c$ *is soft constraint*
  then
    drop c
  else
    output "FAILURE";
    break;
  end
  P=(X,D,C-{c})
end
return P

Algorithm 4: Overconstraint Problem Relaxation

Chapter 4

Proposed Methodology for the Discrete Controller: Integrated Scheme

4.1 Reconfigurable Coordination in the Discrete Time

Let's briefly summarize the tools we have so far analysed to our original end, namely *connectivity-preserving reconfigurable coordination based on a given specifications set.*

We assume that the network of robots in leader-follower scheme is initially deployed in an unknown workspace. Initial topology of the network satisfies the local connectivity specifications of followers. Moreover, a path is given to be followed by the leader. At time instant k, followers should decide about their next positions on workspace (at $k+1$) so as to keep satisfying their specifications set.

Firstly, we presented a distributed algorithm for extracting the followers next positions whenever a solution to the problem with the original specifications set exists. Otherwise, a signal is communicated to the network indicating that the original problem has no solution (or, equivalently, network cannot follow the leader's motion maintaing its current connectivity status) and reconfiguration has to take place. For the last case, we developed a centralized reconfiguration approach that iteratively drops local connectivity specifications (while checking that they do not violate global connectivity), until reaching a solution - in other words a less-connected new configuration of the system that allows system's motion in the direction of the leader.

In more details, we reduced the problem to a well-studied problem from the area of Artificial Intelligence, the Distributed Constraint Satisfaction Problem (DCSP). During that reduction, for reasons of safety, local connectivity specifications set should be augmented by a set of collision avoidance specifications among agents.

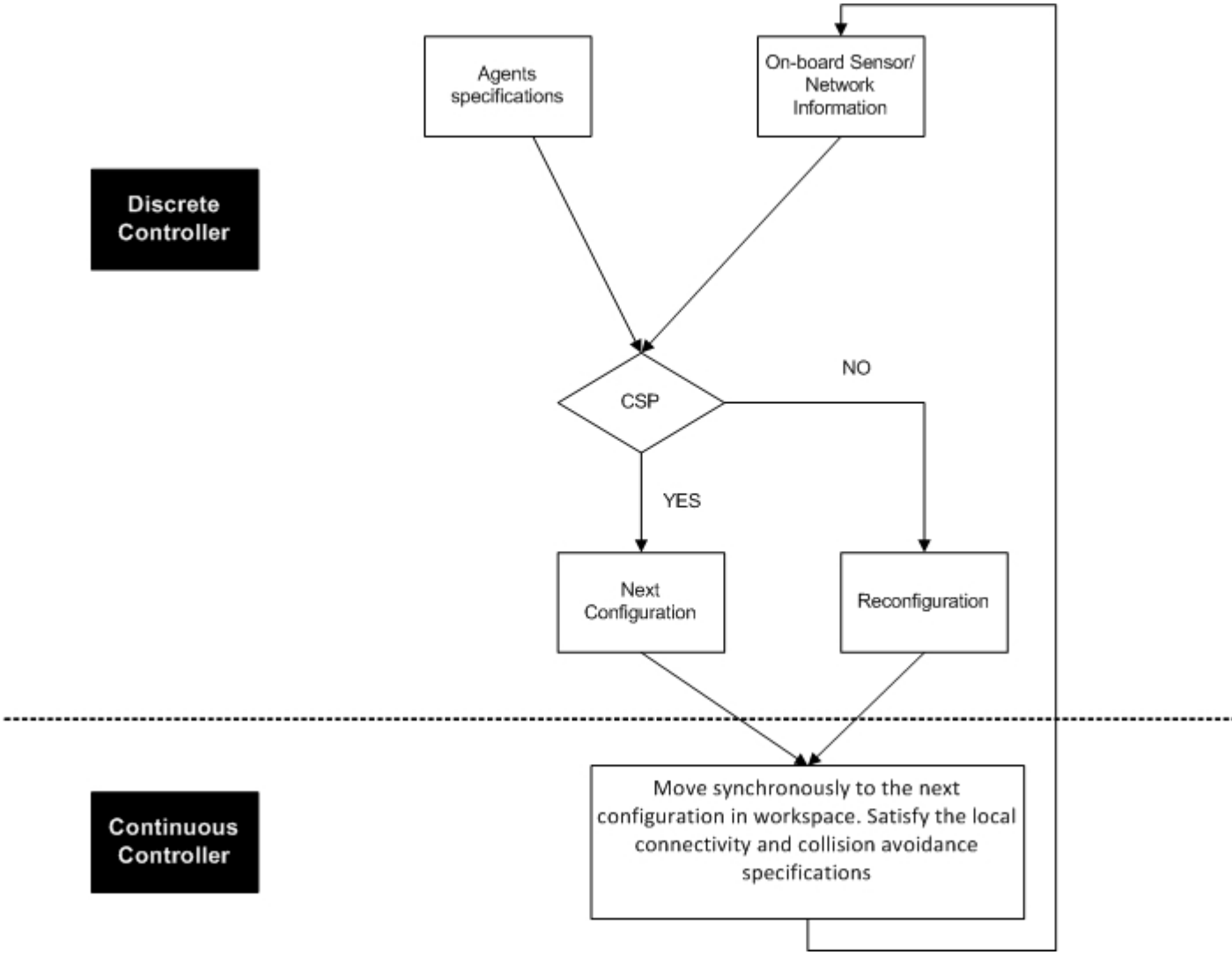


Figure 4.1: Integrated architecture of the proposed scheme. This flow of computations is iterated for each agent in distributed manner. However, the process of reconfiguration is centralized for the whole network.

4.1.1 Synchronisation

Decisions about each agent's next coordinates should have been completed before moving to the continuous part of the controller. Each agent converges asynchronously to its solution during the execution of asynchronous backtracking for the DCSP and a signal of termination should be communicated to the whole system, when all agents have found their solution. In case of reconfiguration, an indicative signal should be propagated. In this case, computations will proceed in a centralised scheme.

4.1.2 The Role of the Leader

The role of the leader is simply to decide about which direction the network will move to. Leader's path is designed *a priori*, via a method we do not take into futher consideration - e.g. it could be by implementing A* in a prior estimation, when the task is of the simple form "go to the target", or through another algorithm for a more sophisticated motion task (expressed in LTL, for instance). In case its initial plan is implementable on the workspace, the leader does not have to do any futher computation, apart from synchronising its motion with the followers. Moreover, we are not strictly attached to a scheme with a unique or a specific leader. This is to say that, depending on the task, system can be easily adapted to have more than one leaders or change leader throughout task execution.

4.1.3 Integration of Obstacle & Collision Avoidance

Obstacle avoidance is straightforwardly achieved through eliminating from the domain of values for each agent those regions of the workspace that are occupied by obstacles. Obstacle sensing is performed in real-time via on-board sensors and it is sufficient for our scheme that obstacles are discovered when agents move to neighboring regions in the workspace. Moreover, for our discrete controller we assume that workspace is not changed during the continuous part of motion. However, system could be subject to dynamic changes during the execution of the discrete controller (before agents have converged to a solution).

Collision avoidance is considered to be achieved simplistically if any two agents of the system do not coexist at the same region of the workspace. Collision avoidance is inserted to the system as a set of additional individual specifications. Concretely, apart from its local connectivity specifications, is given a set of collision avoidance specifications with agents that are reachable in two steps (since two agents reachable in two steps can collide should they decide both to move in opposite directions). This fact slightly increases the connectedness of the specifications graph taken into account while solving DCSP.

4.1.4 Specifications Removal

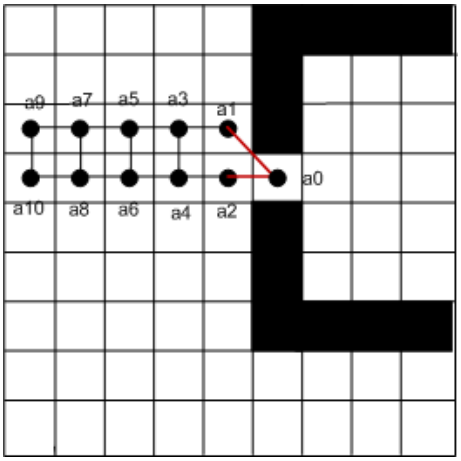
Removal of specifications is performed when the system enters the phase of a centralized reconfiguration. Edges that can be removed are edges of the specifications graph that correspond to local connectivity constraints and are not critical for the global connectivity. This set of "soft connectivity" specifications is adjusted throughout reconfiguration, since the iterative removal of edges changes the sets of hard and soft connectivity specifications. However, it always converges to a relaxed solution of the coordination problem, if the workspace is connected. Furthermore, as we have already mentioned, agents can decide whether an edge is critical in a distributed way, albeit the decision about which edge will be dropped is taken centrally.

4.2 Example

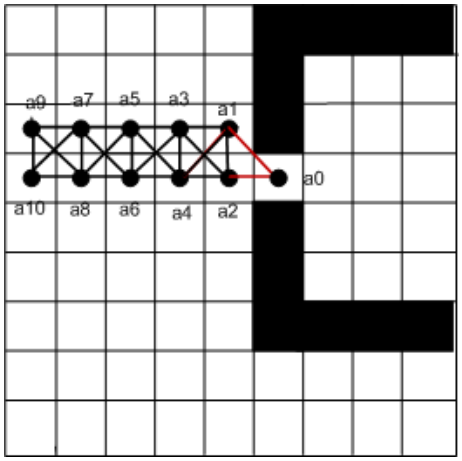
We track the execution of the algorithm at an example involving reconfiguration.

At first, we give some notes about the specific workspace. Worskpace is considered discretized via a rectangular tesselation. Each cell has 8 neighbors, namely its up, down, left, right and diagonal cells. Each agent can move omnidirectionally - thus, given current coordinates of the agent, its subsequent position will be one of its neighboring or the agent will remain in its current cell. On-board sensors range will also be one cell. Therefore, obstacles are discovered when agent moves to a neighboring cell and communication can take place only with neighboring agents.

DCSP solver terminates by indicating that no solution exists. Upon receival of that fact system extracts a MUC. Reconfiguration on the level of specifications is taking place. System characterises both $a_0 - a_1$ and $a_0 - a_2$ edges as soft and drops one of them at random (if no further optimization criteria exist). Let's assume that edge $a_0 - a_1$ is dropped. Then DCSP solver runs for the new relaxed version of original problem and provides the solution displayed in the following.



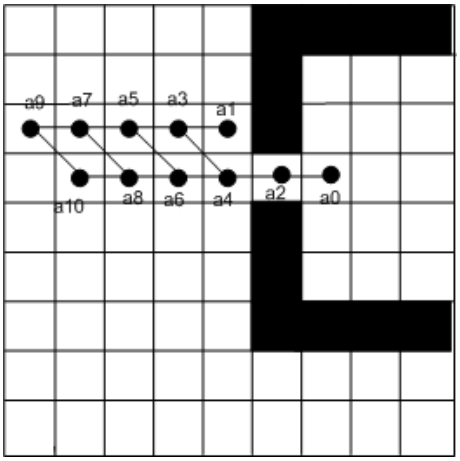
(a) t=0: Local Connectivity Specifications Graph



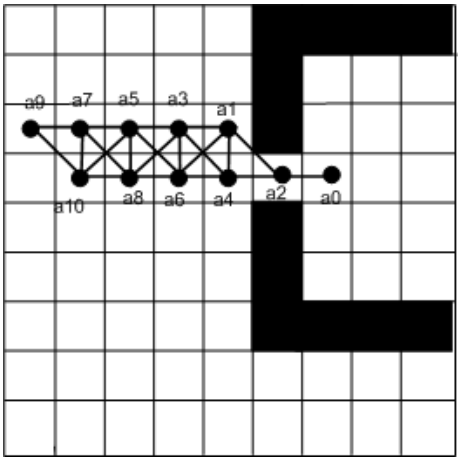
(b) t=0: Topological Graph

SYSTEM'S SPECIFICATIONS	
AGENT	**ATOMIC SPECIFICATIONS**
a1	Collision avoidance: a0,a2,a3,a4,a5,a6 (Hard) Local connectivity: a0,a3 (Soft)
a2	Collision avoidance: a0,a1,a2,a4,a5,a6 (Hard) Local connectivity: a0,a4 (Soft)
a3	Collision avoidance: a0,...,a8 (Hard) Local connectivity: a1,a4,a5 (Soft)
a4	Collision avoidance: a0,...,a 8 (Hard) Local connectivity: a2,a3,a6 (Soft)
a5	Collision avoidance: a1,...,a10 (Hard) Local connectivity: a3,a6,a7 (Soft)
a6	Collision avoidance: a1,...,a10 (Hard) Local connectivity: a4,a5,a8 (Soft)
a7	Collision avoidance: a3,... ,a10 (Hard) Local connectivity: a5,a8,a9 (Soft)
a8	Collision avoidance: a3,...,a10 (Hard) Local connectivity: a6,a7,a10 (Soft)
a9	Collision avoidance: a5,...,a10 (Hard) Local connectivity: a7,a10 (Soft)
a10	Collision avoidance: a5,...,a9 (Hard) Local connectivity: a8,a9 (Soft)

MUC	
AGENT	**ATOMIC SPECIFICATIONS**
a1	Collision avoidance: a0,a2,a3,a4,a5,a6 (Hard) Local connectivity: a0,a3 (Soft)
a2	Collision avoidance: a0,a1,a2,a4,a5,a6 (Hard) Local connectivity: a0,a4 (Soft)



(c) t=1:Local Connectivity Specifications Graph



(d) t=1:Topological Graph

Figure 4.2: Task subject to specifications reconfigurations. At t=0, MUC is extracted - comprised from indicated local connectivity and respective collision avoidance specification. On the left local connectivity specifications is displayed, while on the right topological graph is displayed. At t=1, we depict network graphs associated with found solution.

In the next step, initially we will try again to find a solution to the original problem. After running DCSP solver, we will deduce that again no such solution exist. A new set of MUCs will be found and the system will reach a relaxed version of the original problem by dropping soft specifications within them. Formulating identically the coordination problem, may seem a vague approach, since when the system enters a region of dense obstacles it is plausible to suppose that solutions after reconfiguration will be increasingly relaxed. However, starting searching on the basis of the previous relaxed instant of CSP will lead necessarily to a sequence of less and less connected discrete configurations of the network. Furthermore, this approach will prohibit *initial connectivity status restoration* when robots exit the regions with obstacles and are led to a free area of the workspace.

4.3 Continuous Controller

The continuous part of motion, namely agents motion between two successive positions defined by the discrete controller, is designed separately. The two basic objectives of continuous controller are:

1. **Collision Avoidance:** Transition from cell to cell is performed without moving to another cell in between.

2. **Synchronisation:** Since decision about next discrete configurations are taken synchronously, agent should synchronously leave their previous cell and arrive at their next target.

Design and philosophy of the continuous controller is based on the notion of Prescribed Performance Control, introduced in [4]. According to the above mentioned objectives, we define appropriate performance functions. By prescribed performance it is meant that the output tracking error should converge to a predefined arbitrarily small residual set, with convergence rate no less than a certain prespecified value, exhibiting maximum overshoot less than a sufficiently small preassigned constant.

The necessity for collision avoidance, combined with a prescribed control based continuous controller implies that some further restrictions regarding the tessellation may be introduced to the workspace. For instance, diagonal motion in a rectangular grid should be ruled out, since it can cause a collision. Thus, 9-rectangle neighborhoods should be restricted to 4-rectangle neighborhoods (containing up, down, left and right cell). A far more appropriate tessellation for our scheme would be an hexagonal one, which will be used for the simulations performed in the next section.

Chapter 5

Results

5.1 Simulations

In this chapter we present the results we received by applying our control scheme on two different platforms. In the first simulation, the continuous part of motion was not integrated, while in the second simulation both discrete and continuous part were implemented.

5.2 Simulation 1

5.2.1 Technical Description

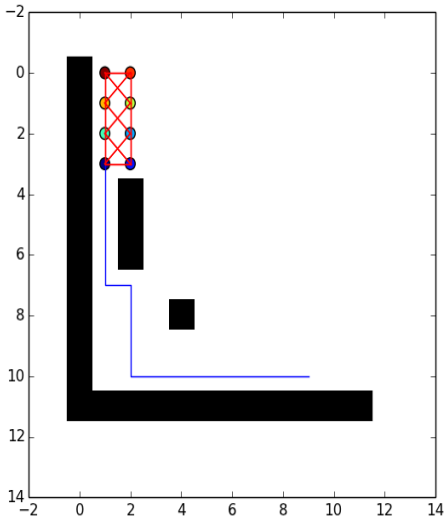
In simulation 1 we consider a workspace in the plain with rectangular tessellation. We assume that agents carry on-board omnidirectional sensors with range equal to $\sqrt{2}$. Thus, local connectivity exists among cells belonging to 9-square neighborhoods. Obstacle sensing is achieved when agents move at cells adjacent to an obstacle. No prior knowledge of the workspace, as well as no memory is assumed. Leader's path is designed *a priori*. Moreover, we admit that robots are holonomic (they can decide to move towards each of the adjacent cells of their current position).

Leader moves on the successive centers of the regions crossed by its path, while followers should obey their local connectivity and collision avoidance specifications, by solving DCSP at each step.

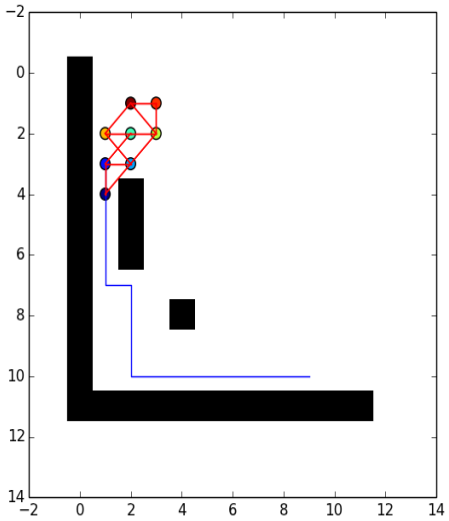
Code was implemented in Python 2.7 and executed in computer with Processor Intel Core i7-2630QM, RAM 4 GB. Execution time of the programm for the whole motion task was very few seconds.

5.2.2 Evaluation

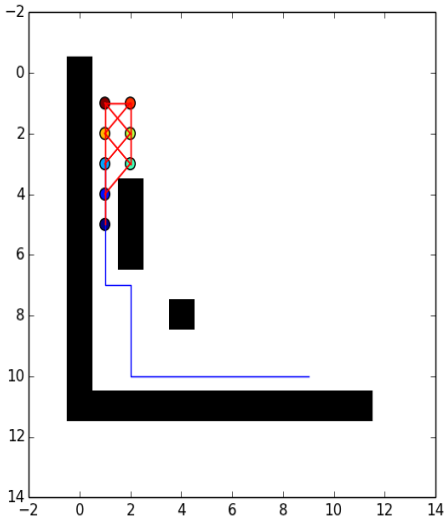
We confirm that the network is able to reconfigure efficiently and robustly its topology in order to carry out the assigned task motion without loss of global connectivity. In particular, we observe that progressively (k=0 until k=4) network extends itself and approaches



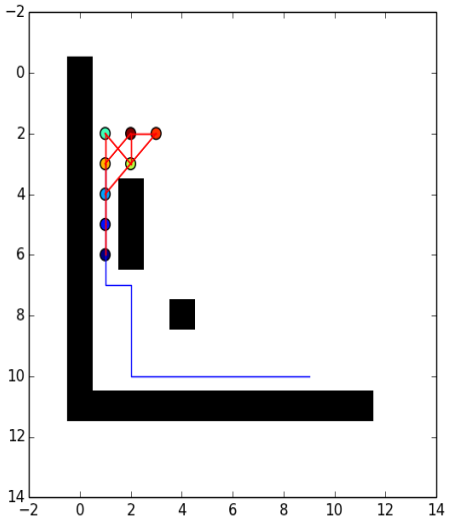
k=0



k=1

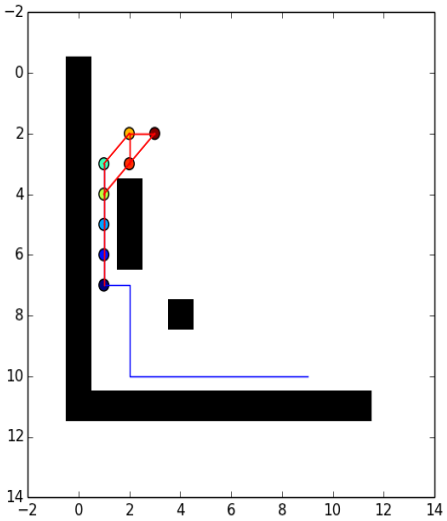


k=2

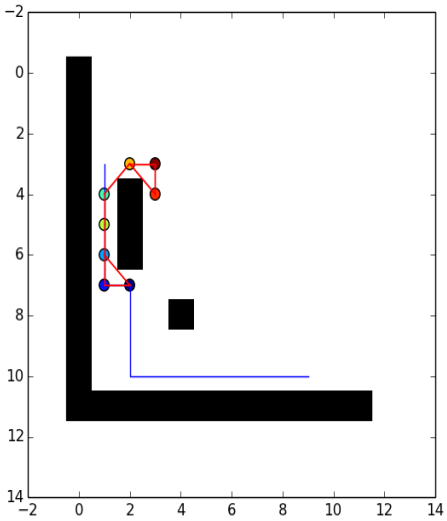


k=3

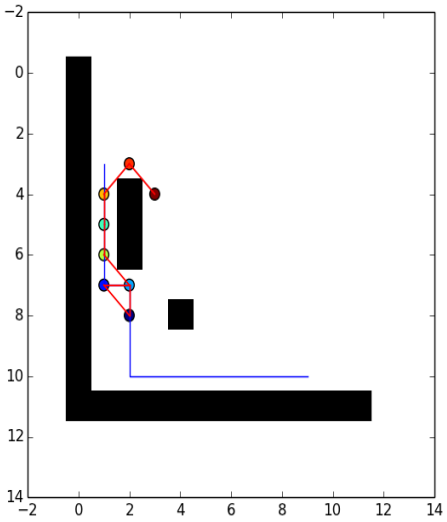
Figure 5.1: Simulations 1.1



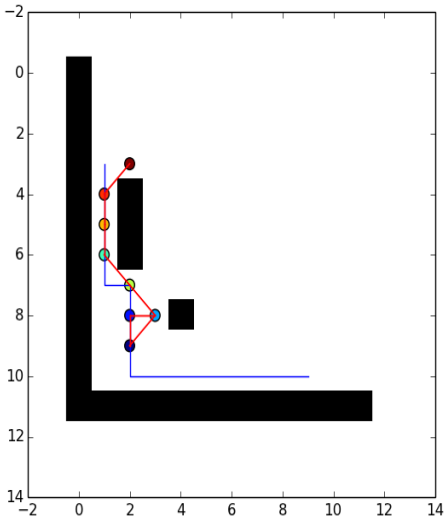
k=4



k=5

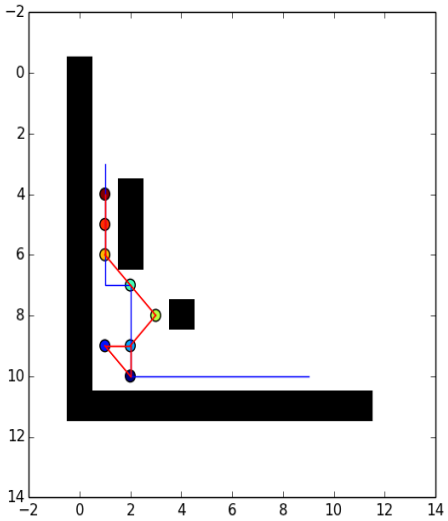


k=6

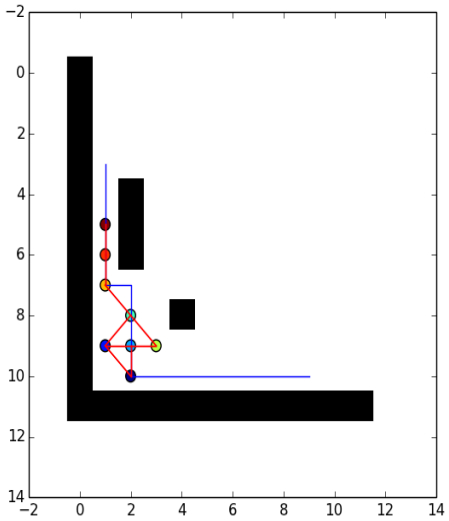


k=7

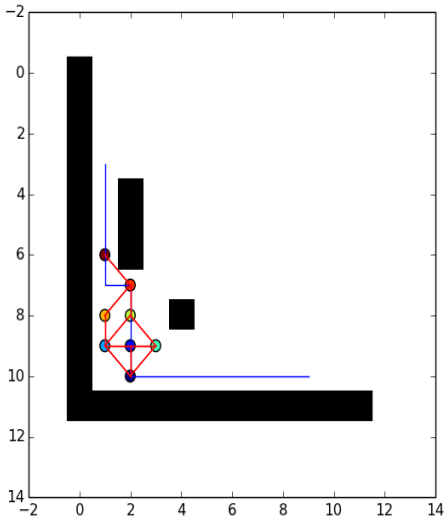
Figure 5.2: Simulations 1.2



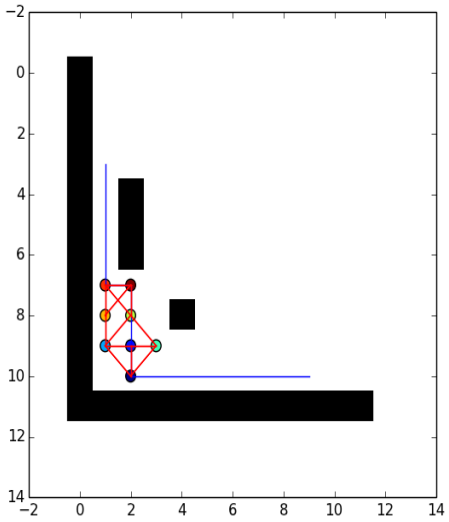
k=8



k=9

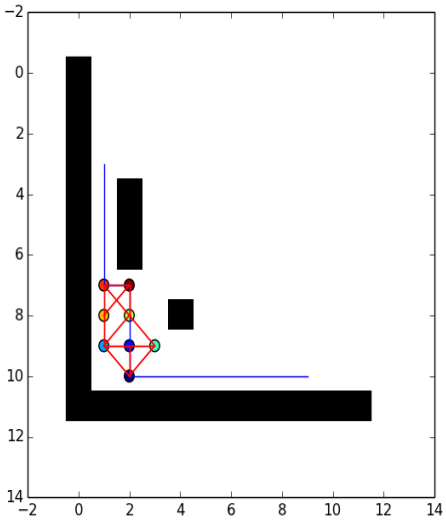


k=10

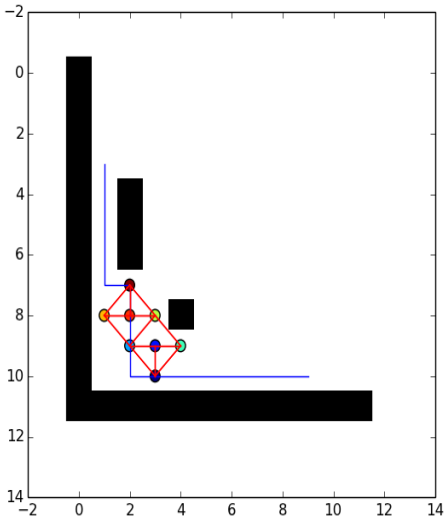


k=11

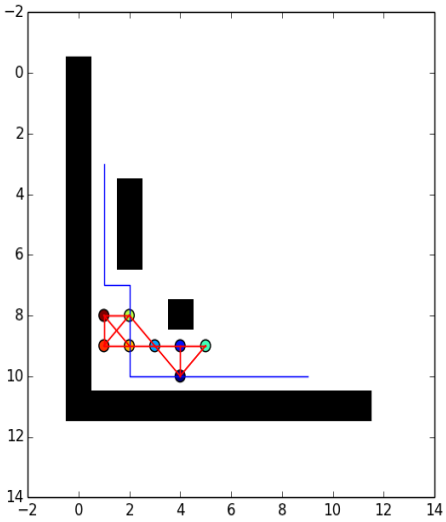
Figure 5.3: Simulations 1.3



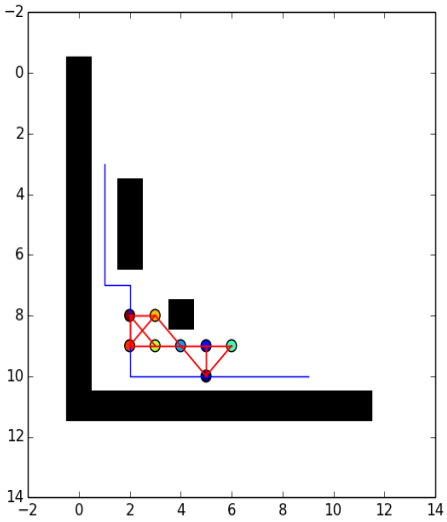
k=12



k=13

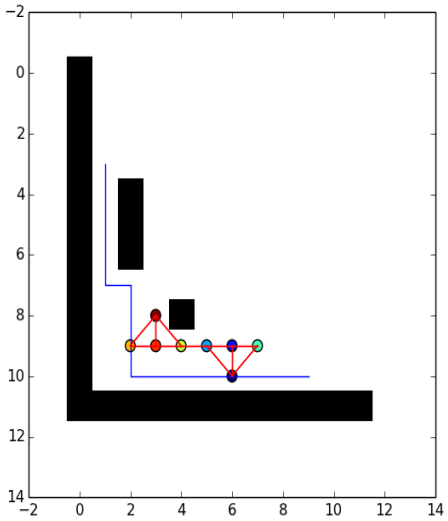


k=14

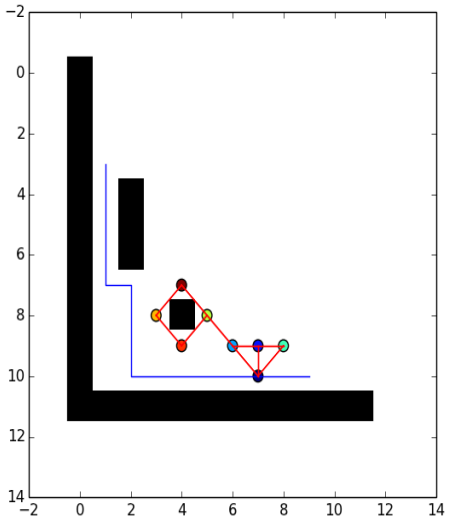


k=15

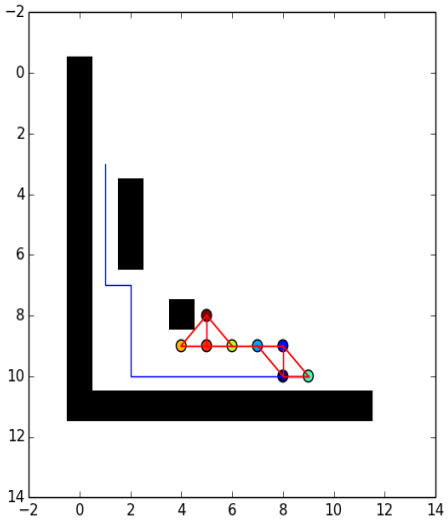
Figure 5.4: Simulations 1.4



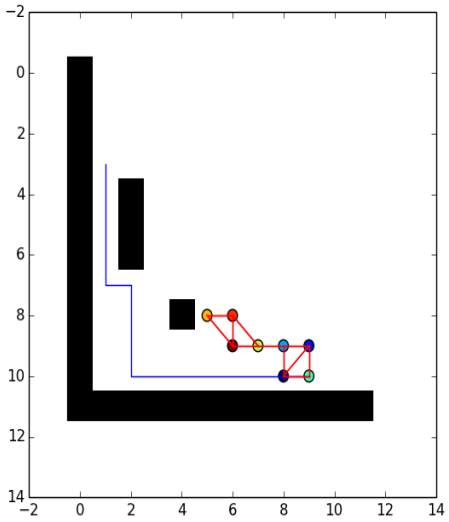
k=16



k=17

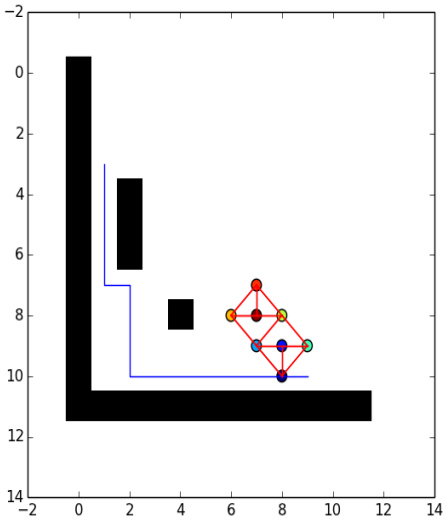


k=18

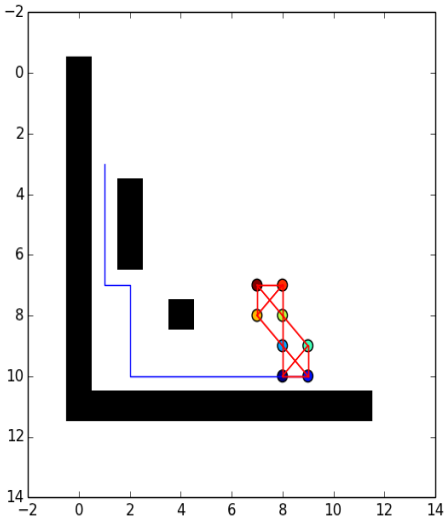


k=19

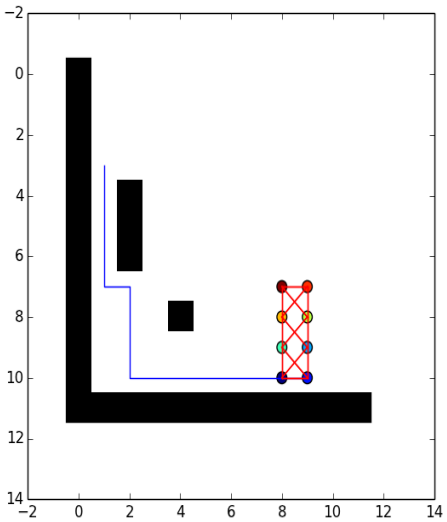
Figure 5.5: Simulations 1.5



k=20



k=21



k=22

Figure 5.6: Simulations 1.6

a near-linear topology in order to pass through the long and narrow corridor created by the obstacles. Subsequently, upon exiting the obstacle, exploiting the discovered free space, network finds a better solution to the original CSP and increases connectivity(k=5 until k=12). Then, the network encounters the square obstacles and extends itself again, succeeding to procceed in leader's direction (k=13 until k=14). At this stage, we can observe that the system avoids square obstacle in a rather counter-intuitive way, since an agent decides to move through the empty space on the upper side of the obstacle (see configuration at k=17), while the others move through the lower side of the obstacle. When leader reaches its terminal coordinates, instead of aborting the task, we keep moving the followers until initial connectivity status of the multi-robot system is restored (k=18 until k=22).

Although the discrete controller produces allowable instances for network configuration, there exists a number of issues tricky to be tackled by the continuous controller, due to the specific tessellation. For instance, we have to exclude swaps (mutual interchange of positions) between adjacent agent, since the continuous part of this transition cannot be handled by a prescribed performance control with guarantee of collision avoidance, and thus reach a more rigid scheme. This is an evidence that we should look for an even better tessellation. Hexagonal grid is more appropriate and is the basis for the discretization of workspcae selected in simulations of the next section.

5.3 Simulation 2

5.3.1 Technical Description

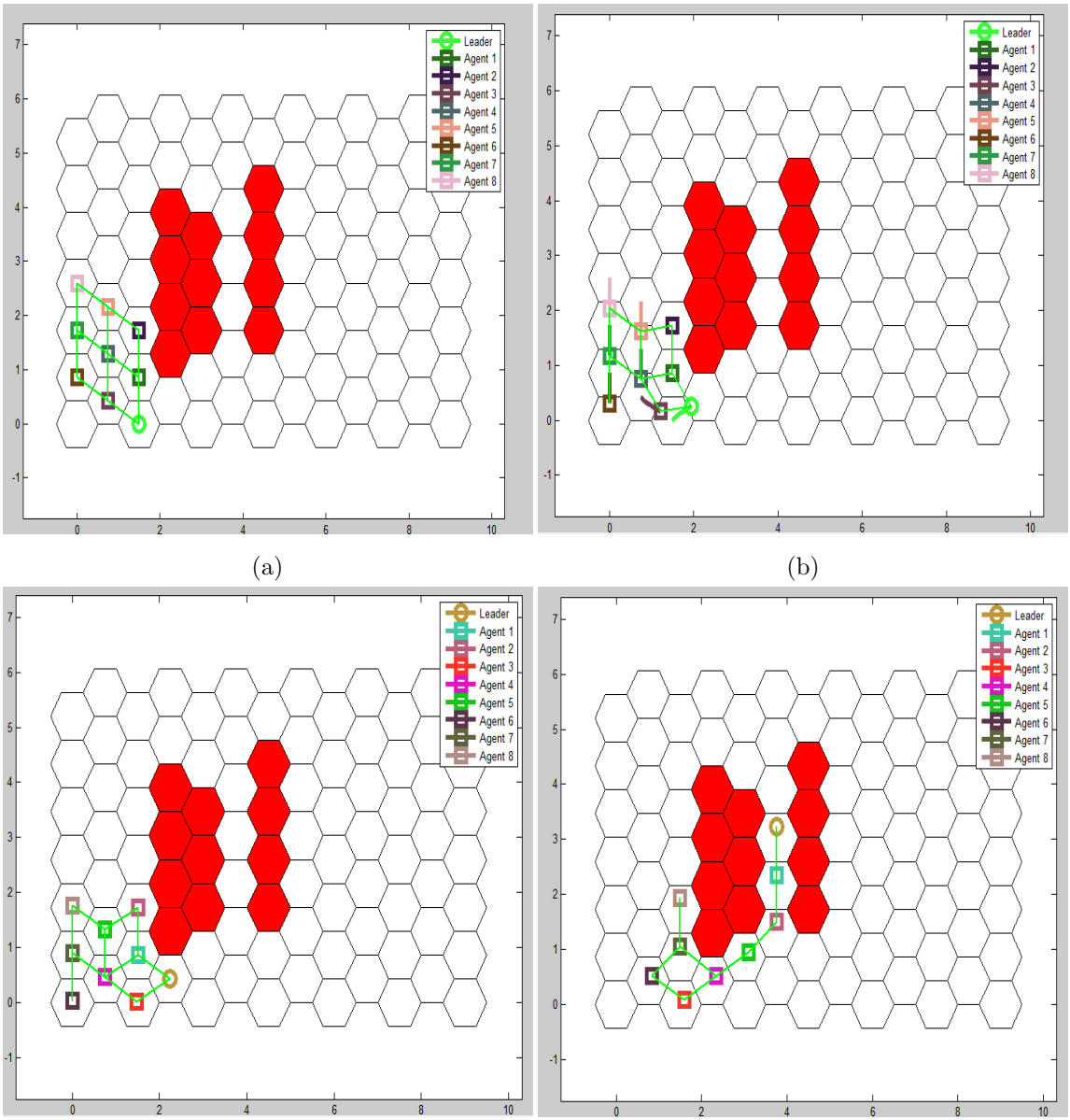
In second simulation we use a hexagonal tessellation of the workspace. The charactristics of robots are identical to the ones of previous simulation. The only difference is that at the current tessellation we have 6-hexagon neighborhoods.

System is initially globally connected and is deployed in the unknown workspace. Leader is assigned a motion task, consisting in following a predefined trajectory.

Discrete controller was implemented in Python as described in first simulation, while for integration with the continuous part and plotting we implemented code in MATLAB (R2011a).

5.3.2 Evaluation

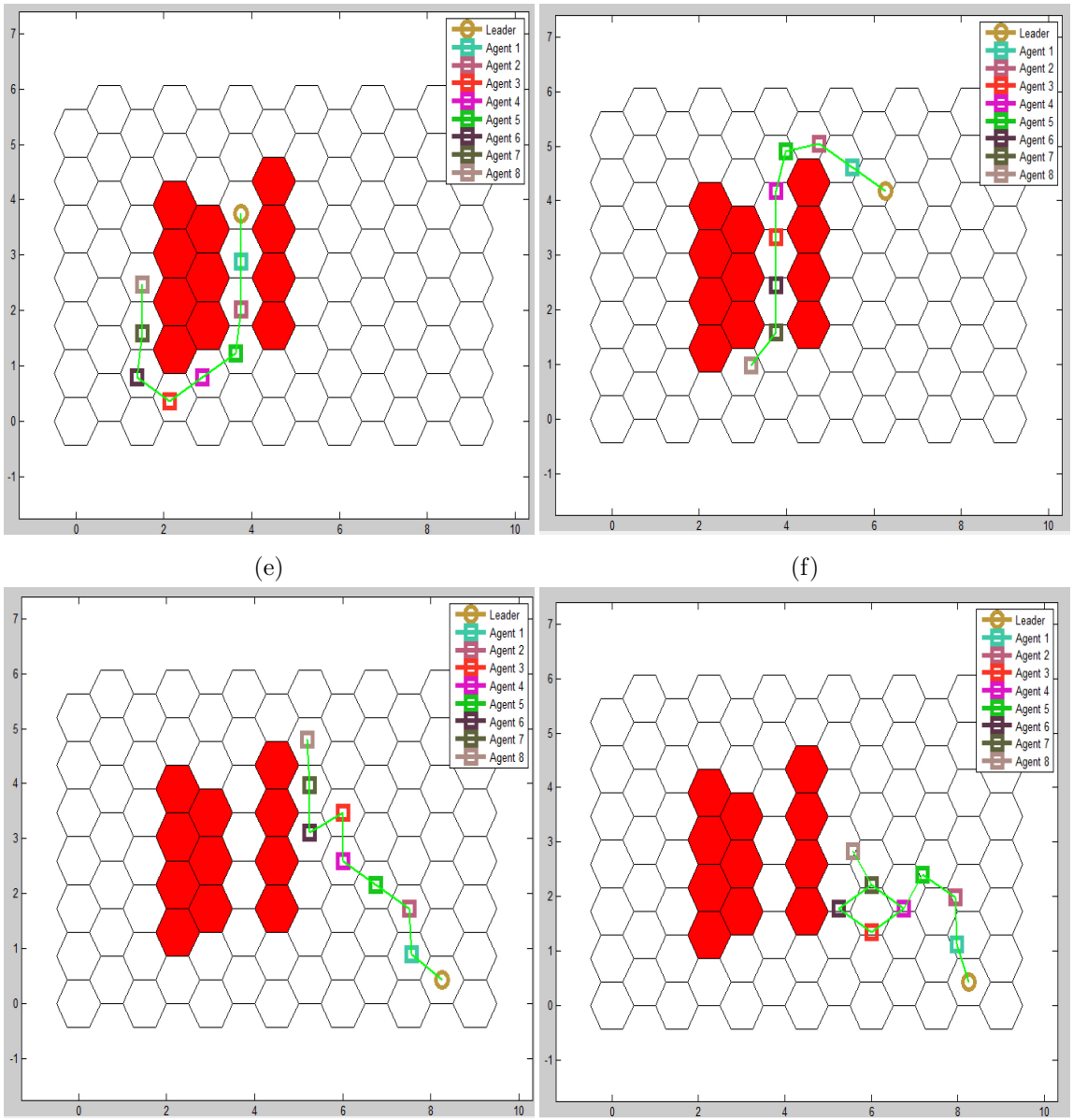
While passing through the narrow corridor between obstacles, the initially connected multi-robot system tends gradually to a chain-like formation, by breaking local connectivity relationships (see subfigures (a)-(f)). Subsequently, the system restores progressively initial connectivity status (see subfigures (g)-(j)). A nice attribute of the controller in the specific example, is that the ordering of the agents in linear formation coincides with the enumeration of them in initial configuration. Furthermore, regarding the continuous



(a) (b)

(c) (d)

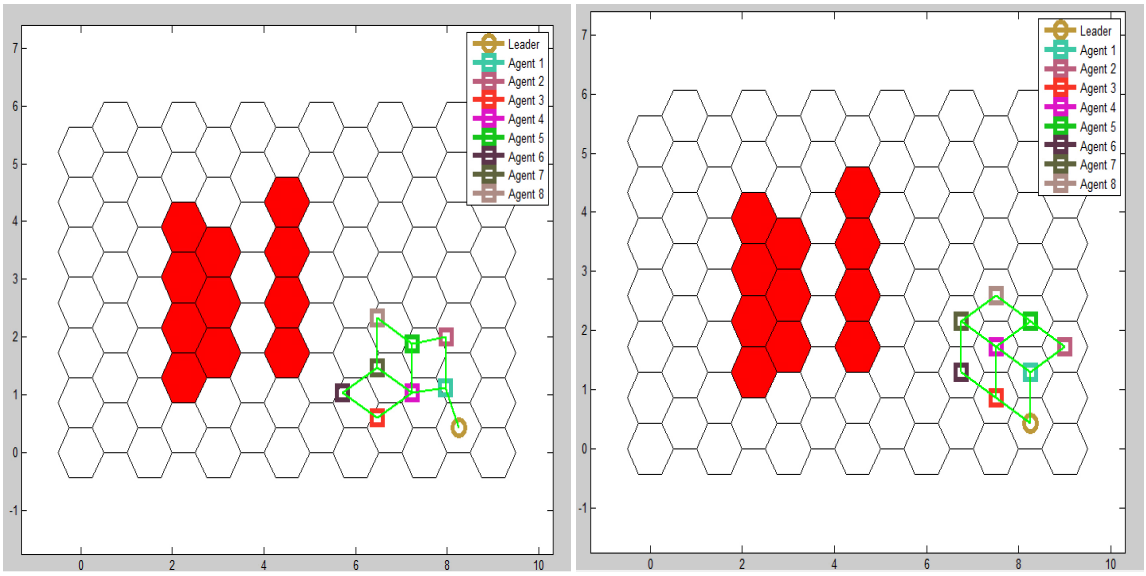
Figure 5.7: Simulations 2.1



(e) (f)

(g) (h)

Figure 5.8: Simulations 2.2



(i) (j)

Figure 5.9: Simulations 2.3

part, we notice that agents enter synchronously their new hexagon (which implies that connectivity is not lost during motion from cell to cell).

Chapter 6

Future Research

In the following we propose some directions for further research to the problem we dealt with. We should mention here that, to our knowledge, work on the boundaries between control theory and artificial intelligence is arguably limited, thus we are inclined to believe that many aspects of the problem worth to be further explored.

- **Further Decentralization: Decentralization of Reconfiguration**

  We have seen that Yokoo et al. decentralized algorithm can determine in finite time if the original DCSP has solution. Otherwise, we run a centralized reconfiguration procedure. Although the process of reconfiguration is centralized, in general we observe that, regarding multi-robot applications, MUCs correspond to local cores within the network (see for example Figure 3.6). Thus, we have certain indications that elimination of MUCs can be achieved with local information. Extraction of MUCs is also centralized. However, a heuristic preprocessing step can be incorporated in DCSP solver by indicating which agents face the most conflicts while searching for a solution the problem. This mechanism could allow us to exclude some (hopefully most of) agents of the network from the reconfiguration procedure.

- **The Role of the Leader / Design of Leader's Path / Early Detection of Deadlocks**

  Our scheme is essentially decoupled from the methodology of designing leader's path. Concretely, we assume that leader's trajectory design is carried out by another controller and is based on the complexity of motion task that should be executed by our multi-robot system. However, *leader's next step coordinates* should be given in every run of our discrete controller, since they define the direction towards which the network moves. Our scheme allows changes of initially designed leader's trajectories. For instance, if leader's sensor has range greater than 1-step ahead and detects an obstacle crossing its trajectory, leader can stay at its coordinates and design a new free of obstacles path.

  Moreover, restorage of initial connectivity status in free regions of the workspace can be achieved by iterating our scheme few times with leader paused at its present

coordinates. A leader able to sense broad regions in the workspace could manage to efficiently allow connectivity restorage when possible.

- **Multi-leader schemes/ Dynamic selection of leader**

  Although our scheme guarantees global connectivity maintenance by adjusting system's connectivity graph to obstacles in every connected space, for specific tasks (e.g. surveillance) employing additional agents as leaders may be useful, since it would allow to the system to be divided in multiple teams. However, in this scenario, leaders trajectories should be very carefully designed and synchronised in order to allow future global connectivity restorage, without collisions.

  Moreover, another attempt to make the model more powerful would be to allow dynamic assignment of the role of leader to the agents. For instance, imagine a system in linear formation blocked in the dead-end of the maze, with leader in front. In that case, a more efficient technique would be to assign the role of leader to the last agent of the system, since this agent is the only one who can move to free cells.

- **Motion in space free of obstacles /Dealing with counter-intuitive motion** Since we have not employed agents with memory, system tries to repeateadly solve identical CSPs when moving in regions with no obstacles - identical instances of CSP can be recognized by comparison of the domains of the variables involved in the problem. In these cases, it would be plausible to add memory to the agents in order to remember their previous solution to their problem and not to seek for alternative solution (this method could also allow formation maintenace when it is possible). Furthermore, due to the nature of the problem we solve (which is essentially a problem of exhaustive search in finite domains) we often see counter-intuitive attributes to the solution we obtain (e.g. two neighbors may not select to move towards the same direction, but towards diagonal direaction without losing connectivity). Thas sort of attributes could be excluded by explicitely coupling decision among neighbors in the network - however, that would pose restrictions to the decentralization of solving CSP.

- **Tessellation**

  Discretization of the workspace should be considered along the kinematics and dynamical model of the agents, the properties of selected continuous controller scheme as well as sensing capabilities of the agents. In particular, for our methodology, integrated with Prescribed Performance Controller for the continuous part we realized that hexagonal grid seem more reasonable than rectangular grids.

- **Extending the Method to 3 Dimensions**

  Although we implemented our scheme on plane workspaces, it is straightforward to expand the methodology for spaces with more dimensions.

- **Minimization of Calls to the Discrete Controller**

  In the proposed scheme discrete and contrinuous controller work sequentially at each step of the cooperative task motion. Namely, in the first place, discrete controller decides about network configuration, by producing an assignment of each agent to its next allowable coordinates. Next, continuous controller supervises the transition of each agents from present to subsequent coordinates. Thus, discrete controller is called at each step of motion.

  A more efficient scheme would be to implement motion in the continuous level and call discrete controller "by need" whenever continuous system "gets stuck" - then discrete controller would be essentially a scheme that supervises reconfigurations. A significant question in that approach would be the following: how would the continuous controller decide that it has got stuck?

Appendix A

Basic Notions from Algebraic Graph Theory

Algebraic Graph Theory ([14]) provides a compact mathematical framework for abstract modeling of multi-agent systems. In this section we recall some basic concepts.

We define a undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{u_1, u_2, ..., u_n\}$ are the vertices and $\mathcal{E}$ are the edges of the graph.

Degree Matrix $\Delta(\mathcal{G})$. The diagonal matrix which contains information about the degree of each vertex of $\mathcal{G}$ - that is, the number of edges attached to each vertex.

Adjacency Matrix $A(\mathcal{G})$. The symmeric matrix which represents which vertices of the graph are adjacent to which other vertices, as follows:

$$|A(\mathcal{G})|_{ij} = \begin{cases} 1, & \text{if } u_{ij} \in \mathcal{E} \\ 0, & \text{otherwise} \end{cases} \quad \text{(A.1)}$$

Laplacian Matrix $L(\mathcal{G})$. $L(\mathcal{G}) := \Delta(\mathcal{G}) - A(\mathcal{G})$. It is a symmetric and positive semidefinite matix - thus having real eigenvalues. Furthermore, from its definition is easy to observe that all rows of the matrix sum up to 0 - consequently, the rank of the matrix is at most $n - 1$. By examining the Laplacian matrix we can derive useful conclusions regarding graph's properties, as implied by the following theorem.

Theorem 1.2. *Assume the following order of Laplacian's eigenvalues :* $0 = \lambda_1(\mathcal{G}) \leq \lambda_2(\mathcal{G}) \leq ... \leq \lambda_n(\mathcal{G})$. *Then graph* $\mathcal{G}$ *is connected iff* $\lambda_2(\mathcal{G}) > 0$.

Bibliography

[1] W. Abbas and M. Egerstedt. Graph topologies for networked systems. In *3rd IFAC Workshop on Distributed Estimation and Control in Networked Systems,Santa Barbara, CA*, Sep 2012.

[2] A. Ajorlou and A.G. Aghdam. Connectivity preservation in nonholonomic multi-agent systems: A bounded distributed control strategy. *Automatic Control, IEEE Transactions on*, 58(9):2366–2371, Sept 2013.

[3] Derya Aksaray, A Yasin Yazicioglu, Eric Feron, and Dimitri N Mavris. A message passing strategy for decentralized connectivity maintenance in agent removal. *arXiv preprint arXiv:1311.0244*, 2013.

[4] Charalampos P Bechlioulis and George A Rovithakis. Robust partial-state feedback prescribed performance control of cascade systems with unknown nonlinearities. *Automatic Control, IEEE Transactions on*, 56(9):2224–2230, 2011.

[5] Calin Belta, Antonio Bicchi, Magnus Egerstedt, Emilio Frazzoli, Eric Klavins, and George J. Pappas. Symbolic planning and control of robot motion [grand challenges of robotics]. *IEEE Robot. Automat. Mag.*, 14:61–70, 2007.

[6] F. Bullo, J. Cortes, and S. Martinez. *Distributed Control of Robotic Networks*. Princeton University Press, 2009.

[7] Yushan Chen, Xu Chu Ding, and Calin Belta. Synthesis of distributed control and communication schemes from global ltl specifications. In *CDC-ECE*, pages 2718–2723. IEEE, 2011.

[8] M.C. De Gennaro and A. Jadbabaie. Decentralized control of connectivity for multi-agent systems. In *Decision and Control, 2006 45th IEEE Conference on*, pages 3628–3633, Dec 2006.

[9] D. Di Paola, R. De Asmundis, A. Gasparri, and A. Rizzo. Decentralized topology control for robotic networks with limited field of view sensors. In *American Control Conference (ACC), 2012*, June 2012.

[10] Dimos V. Dimarogonas and Kostas J. Kyriakopoulos. Decentralized navigation functions for multiple robotic agents with limited sensing capabilities. *Journal of Intelligent and Robotic Systems*, 48(3):411–433, 2007.

[11] Dimos V. Dimarogonas, Savvas G. Loizou, Kostas J. Kyriakopoulos, and Michael M. Zavlanos. A feedback stabilization and collision avoidance scheme for multiple independent non-point agents. *Automatica*, 42(2):229 – 243, 2006.

[12] Xu Chu Ding, M. Kloetzer, Yushan Chen, and C. Belta. Automatic deployment of robotic teams. *Robotics Automation Magazine, IEEE*, 18(3):75–86, Sept 2011.

[13] Ioannis Filippidis, Dimos V. Dimarogonas, and Kostas J. Kyriakopoulos. Decentralized multi-agent control from local ltl specifications. In *51st IEEE Conference on Decision and Control (CDC 2012)*, pages 6235 – 6240, Maui, Hawaii, USA, December 10-13 2012.

[14] C. Godsil and G. Royle. *Algebraic Graph Theory*, volume 207 of *Graduate Texts in Mathematics.* volume 207 of Graduate Texts in Mathematics. Springer, 2001.

[15] Eric Gregoire, Bertrand Mazure, and Cedric Piette. On finding minimally unsatisfiable cores of csps. *International Journal on Artificial Intelligence Tools*, 17(4):745–763, 2008.

[16] S. Grime, H.F. Durrant-Whyte, and P. Ho. Communication in decentralized data-fusion systems. In *American Control Conference, 1992*, pages 3299–3303, June 1992.

[17] Meng Guo and Dimos V. Dimarogonas. Reconfiguration in motion planning of single- and multi-agent systems under infeasible local ltl specifications. In *CDC*, pages 2758–2763, 2013.

[18] Meng Guo, Karl Henrik Johansson, and Dimos V. Dimarogonas. Revising motion planning under linear temporal logic specifications in partially known workspaces. In *ICRA*, pages 5025–5032. IEEE, 2013.

[19] Fred Hemery, Christophe Lecoutre, Lakhdar Sais, and Frédéric Boussemart. Extracting mucs from constraint networks. In *Proceedings of the 2006 Conference on ECAI 2006: 17th European Conference on Artificial Intelligence August 29 – September 1, 2006, Riva Del Garda, Italy*, pages 113–117, Amsterdam, The Netherlands, 2006. IOS Press.

[20] Marius Kloetzer and Calin Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Trans. Automat. Contr.*, 53:287–297, 2008.

[21] Hadas Kress-Gazit. *Transforming high level tasks to low level controllers.* PhD dissertation, University of Pennsylvania, 2008.

[22] Savvas G. Loizou and Kostas J. Kyriakopoulos. Automatic synthesis of multiagent motion tasks based on ltl specifications. In *IN PROCEEDINGS OF THE 43RD IEEE CONFERENCE ON DECISION AND CONTROL*, pages 153–158, 2004.

[23] Mehran Mesbahi and Magnus Egerstedt. *Graph theoretic methods in multiagent networks*. Princeton series in applied mathematics. Princeton University Press, Princeton (N.J.), 2010.

[24] Nathan Michael, Michael M. Zavlanos, Vijay Kumar, and George J. Pappas. Maintaining connectivity in mobile robot networks. In *Experimental Robotics*, pages 117–126. Springer Berlin Heidelberg, 2009.

[25] E. Rimon and D.E. Koditschek. Exact robot navigation using artificial potential fields. *IEEE Transactions on Robotics and Automation*, 8(5):501–518, 1992.

[26] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

[27] Lorenzo Sabattini, Nikhil Chopra, and Cristian Secchi. Decentralized connectivity maintenance for cooperative control of mobile robotic systems. *The International Journal of Robotics Research*, 32(12):1411–1423, 2013.

[28] Jay Wagenpfeil, Adrian Trachte, Takeshi Hatanaka, Masayuki Fujita, and Oliver Sawodny. A distributed minimum restrictive connectivity maintenance algorithm. *Proc. 9th International Symposium on Robot Control, Gifu, Japan*, pages 499–504, 2009.

[29] A.Y. Yazicioglu, M. Egerstedt, and J.S. Shamma. Decentralized degree regularization for multi-agent networks. In *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*, pages 7498–7503, Dec 2013.

[30] Makoto Yokoo, Edmund H Durfee, Toru Ishida, and Kazuhiro Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *Knowledge and Data Engineering, IEEE Transactions on*, 10(5):673–685, 1998.

[31] Michael M. Zavlanos, Magnus B. Egerstedt, and George J. Pappas. Graph-theoretic connectivity control of mobile robot networks. *Proceedings of the IEEE*, 99(9):1525–1540, 2011.

[32] Michael M. Zavlanos and George J. Pappas. Potential fields for maintaining connectivity of mobile networks. *Robotics, IEEE Transactions on*, 23(4):812–816, Aug 2007.

[33] Michael M Zavlanos and George J Pappas. Distributed connectivity control of mobile networks. *Robotics, IEEE Transactions on*, 24(6):1416–1428, 2008.