Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Μηχανολογων Μηχανικων

Εργαστηριο Αυτοματου Ελεγχου

Σχεδιασμός αποκεντρωμένου υβριδικού ελεγκτή για δικτυωμένα πολυπρακτορικά συστήματα με εγγυημένη αποφυγή συγκρούσεων και διατήρηση συνδεσιμότητας

Συγγραφέας:
Κωνσταντίνος Βροχιδης

Επιβλέπων:
Καθηγητής Κωνσταντίνος
Ι. Κυριακοπουλος

Οκτώβριος 2016

Περίληψη

Στην παρούσα διπλωματική εξετάζεται ένα δικτυωμένο πολυπρακτορικό σύστημα ρομπότ, αρχιτεκτονικής οδηγού-ακολούθου, ευρισκόμενο σε έναν επίπεδο χώρο εργασίας παρουσία εμποδίων. Προτείνεται μία αποκεντρωμένη στρατηγική αναδιάταξης των προδιαγραφών συνδεσιμότητας και σχηματισμών η οποία εγγυάται τη σύγκλιση του συστήματος στο επιθυμητό σημείο. Πιο συγκεκριμένα, διαμορφώνεται ένας ελεγκτής βασισμένος στην φιλοσοφία των Αποκεντρωμένων Συναρτήσεων Πλοήγησης ο οποίος περιγράφει επαρκώς τους περιορισμούς ασφαλείας καθώς και τους στόχους του συστήματος. Ωστόσο, η εμφάνιση ανεπιθύμητων σημείων ισορροπίας δεν μπορεί να αποκλειστεί. Σε τέτοιες περιπτώσεις χρησιμοποιείται ένας διανεμημένος διακριτός ελεγκτής ο οποίος επιχειρεί να λύσει ένα πρόβλημα Διανεμημένης Ικανοποίησης Περιορισμών σε μία τοπική διαμέριση Voronoi. Η πληροφορία που προκύπτει χρησιμοποιείται για την απαραίτητη αναδιάταξη ώστε το σύστημα να προχωρήσει περαιτέρω προς τον στόχο του. Τελικά, το σύστημα είτε συγκλίνει στο επιθυμητό σημείο, είτε αποκτά τοπολογία δένδρου. Στην δεύτερη περίπτωση το σύστημα χρησιμοποιώντας έναν άλλο ελεγκτή βασιζόμενο στην τεχνική της Προδιαγεγραμμένης Απόδοσης συγκλίνει αποδεδειγμένα. Τα θεωρητικά ευρήματα επιβεβαιώνονται μέσω προσομοιώσεων.

National Technical University of Athens

School of Mechanical Engineering

Control Systems Lab

A decentralized hybrid control scheme for networked multi-agent systems with guaranteed collision avoidance and connectivity maintenance

Author:
Constantinos Vrohidis

Supervisor:
Professor Kostas J. Kyriakopoulos

October 2016

Acknowledgements

First and foremost, I would like to thank my supervisor, Professor Kostas J. Kyriakopoulos for giving me the opportunity to be a member of his lab and for providing me with his deep insight.

Dr. Charalampos Bechlioulis had an immense contribution to the present thesis. I deeply thank him for that, but foremost for always being willing to share his knowledge and expertise in addition to the odd... riddle.

I am also indebted to Panagiotis Vlantis for the numerous interesting and enlightening conversations he shared with me. His scientific integrity is something I look up to.

As a member of the Control Systems Lab, I have had the pleasure of interacting with a number of remarkable individuals.

Chris Mavridis, whom by now I consider a friend, was always helpful. For that I thank him.

I also want to thank Dr. George Karras and Dr. Panos Marantos for creating a positive atmosphere.

Michael, George, Giorgos, Nikos, Shahab, Aris, Zisis, Babis and Chris. Thank you all guys. It's been a pleasure.

To all my friends; I will make it up to you.

Finally, I would like to express my deepest gratitude to my family for always standing by me, reminding me who I want to be.

Abstract

In the present thesis, we consider a networked multi-robot system operating in an obstacle populated planar workspace under a single leader-multiple followers architecture. We propose a decentralized *reconfiguration strategy* of the set of connectivity and formation specifications that assures convergence to the desired point, while guaranteeing global connectivity. In particular, we construct a low-level Decentralized Navigation Functions based controller that encodes the goals and safety requirements of the system. However, owing to topological obstructions, stable critical points other than the desired one may appear. In such case, we employ a high-level distributed discrete procedure which attempts to solve a Distributed Constraint Satisfaction Problem on a local Voronoi partition, providing the necessary reconfiguration for the system to progress towards its goal. Eventually, we show that the system either converges to the desired point or attains a tree configuration with respect to the formation topology, in which case the system switches to a novel controller based on the Prescribed Performance technique, that eventually guarantees convergence. Finally, simulation studies verify the efficacy of the approach.

Contents

1 Introduction

Multi-agent systems 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 systems relies on efficient information exchange and coordination between the members of the team. More specifically, their intriguing feature consists on the fact that each agent makes decisions solely on the basis of its local perception of the environment, which has also been observed in many biological systems [1]. Thus, a challenging task is to design the decentralized control approach for certain global goals in the presence of limited information exchanges. In this direction, drawing some enlightenments from biological observations, distributed cooperative control of multi-agent systems has received considerable attention during the last two decades (see the seminal works [2, 3, 4, 5, 6, 7] for example). In particular, the leader-follower scheme, according to which the following agents aim at reaching a consensus with the leader's state, employing only locally available information, has become very popular, since in the absence of any central control system and without global coordinate information, following a leader is an accountable motivation.

1.1 Literature Review

In [8] the authors address the formation stabilization problem in the presence of point obstacles with connectivity maintenance for acyclic communication graphs using decentralized navigation functions. In [9] flocking with obstacle avoidance is examined using a graph theoretic approach that allows modelling of split/rejoin manoeuvres. Split and join manoeuvres based o a Receding Horizon Approach and a switching law are also discussed in [10]. In [11] a navigation-like function controller is utilize in conjunction with constrained-based programming to prioritize between formation specifications satisfaction and convergence to the destinations in the presence of obstacles. In [12, 13] Prescribed Performance Control methods are used to address the connectivity maintenance and formation problem, respectively. In [14] consensus and formation with connectivity maintenance and bounded control inputs is addressed. Finally, the authors of [15] derive a navigation functions inspired controller that preserves line-of-sight constraints.

2 Problem Formulation

2.1 System Description

We consider a networked system of $n \in \mathbb{N}$ robotic agents, indexed by the set $\mathcal{V} \triangleq \mathbb{N}_{\leq n}$. The first agent is the *leader* while the remaining $n-1$ agents are *followers*. The agents are moving in a subset $\mathbb{W}$ of $\mathbb{R}^2$, governed by the following kinematics:

$$\dot{x}_i(t) = u_i(t), \quad i \in \mathcal{V}, \tag{2.1}$$

where $x_i(t) \in \mathbb{W}$ and $u_i(t) \in \mathbb{R}^2$ denote the position and control input vectors of agent i at time $t \in \mathbb{R}_{\geq 0}$, respectively. The overall system state $x \in \mathbb{R}^{2n}$ is the concatenation of the individual states of the agents. Accordingly, we define the overall system state x as:

$$x \triangleq \begin{bmatrix} x_1^T & \dots & x_n^T \end{bmatrix}^T \in \mathbb{R}^{2n}. \tag{2.2}$$

For every $i \in \mathcal{V}$ we define,

$$\bar{x}_i \triangleq [x_1^T \dots \; x_{i-1}^T \; x_{i+1}^T \dots \; x_n^T]. \tag{2.3}$$

Every agent $i \in \mathcal{V}$ is a robot and thus possesses a physical substance which, in the context of this thesis, will be identified with a disk of radius ρ_{α_i} centered at x_i. We now make the following assumption:

Assumption 1 *All the agents have the same radius henceforth denoted by ρ_α, i.e., $\rho_{a_i} = \rho_a$, $\forall i \in \mathcal{V}$.*

Then, at each time $t \in \mathbb{R}_{\geq 0}$, agent i occupies the set $\mathbb{B}(x_i, \rho_\alpha) \triangleq \{z \in \mathbb{R}^2 : \|z - x_i\| \leq \rho_\alpha\}$.

2.1.1 Sensing Model

All agents are assumed to possess the ability to sense the environment. In particular, every agent is able to sense other entities that are sufficiently close. To make the previous statement mathematically precise, let us identify each entity e with some subset E of $\mathbb{W}$.

Definition 2.1 *The* sensing distance *of an agent $i \in \mathcal{V}$ from a subset E of $\mathbb{W}$, denoted by $d(x_i, E)$ is given by the equation:*

$$d(x_i, E) \triangleq \inf_{z \in E} \|z - x_i\|. \tag{2.4}$$

By *sensing* an entity e, we define the act of acquiring all the necessary information for fully determining the structure of the corresponding set E. For our purposes, we will consider that every agent $i \in V$ can *sense* any entity e if $d(x_i, E) \leq \rho_s$. Then, ρ_s is called the *sensing radius*, with a common value for all agents.

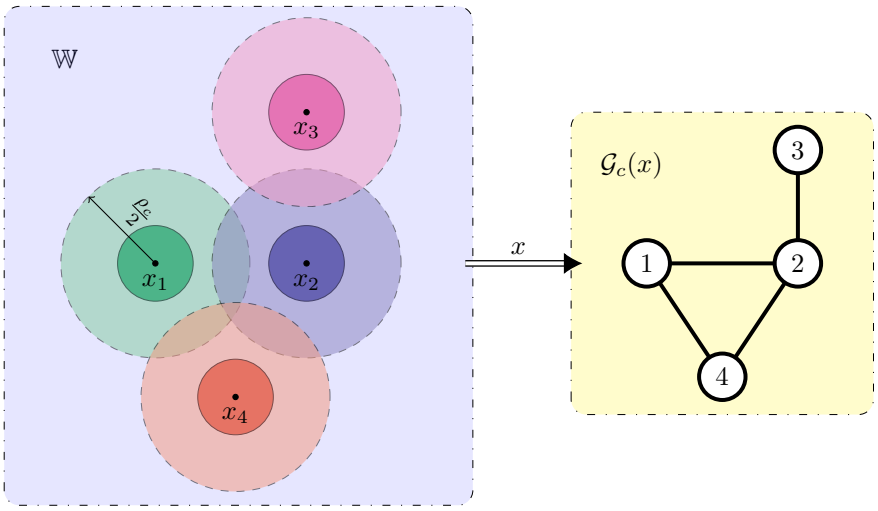


Figure 2.1: A spatial instance of a system comprising of 4 agents and the induced communication graph.

Finally, two entities will be considered separate and will be identified as such, as long as no inclusion on the corresponding set holds. This is a reasonable model since we will only be working with entities that are disk-shaped. Consequently, each entity can be fully described by its center and radius and separate identification of intersecting entities is trivial.

2.1.2 Communication Model

As already mentioned, the multi-agent system under consideration is additionally a networked system. This means that agents are equipped with communication capabilities. Inter-agent communication is considered bidirectional and is governed by one simple rule, namely:

For every pair of agents $i, j \in \mathcal{V}$, with $i \neq j$, *direct* communication between the agents at time t is possible if and only if $\|x_i(t) - x_j(t)\| \leq \rho_c$

The parameter ρ_c is called the *communication radius*. The induced communication network can be mathematically abstracted as a *communication graph*.

Definition 2.2 *The communication graph $\mathcal{G}_c(x(t)) = (\mathcal{V}, \mathcal{E}_c(x(t)))$ is a proximity graph[16], where the set of edges at each time instant t is defined as:*

$$\mathcal{E}_c(x(t)) = \{(i,j) \in \mathcal{V} \times \mathcal{V} \ : i \neq j,\ \|x_i(t) - x_j(t)\| \leq \rho_c\}.$$

Let us now present a simple example illustrating the aforementioned ideas.

Example 2.1. Consider a system consisting of $n = 4$ agents, in a spatial configuration presented in Figure 2.1. The opaque disks have a radius equal to half the communication radius. Thus, two agents can establish direct bidirectional communication as long as the corresponding disks intersect. The *communication graph* $\mathcal{G}_c(x)$ corresponding to the spatial configuration is illustrated on the right. In particular, the set of vertices is $\mathcal{V} = \{1, 2, 3, 4\}$ and the set of edges denoting direct communication is $\mathcal{E}_c(x) = \{(1, 2), (1, 4), (2, 3), (2, 4)\}$.

As in [17], we make the following assumption regarding the values of the communication radius ρ_c and the sensing radius ρ_s.

Assumption 2 *The values of the communication radius ρ_c and of the sensing radius ρ_s coincide, i.e., $\rho_c = \rho_s$.*

2.2 Workspace Description

2.2.1 Definitions

We will now proceed with giving a short description of the workspace set $\mathbb{W}$ where the agents are operating. In particular the *workspace* is defined as the closed disk of radius $\rho_w - \rho_\alpha$, i.e.,

$$\mathcal{W} \triangleq \{q \in \mathbb{R}^2 : \|q\|_2 \leq \rho_w - \rho_\alpha\}. \tag{2.5}$$

Additionally, the workspace is filled by $m \in \mathbb{N}$ disk-shaped *obstacles* indexed by the set $I_o \triangleq \mathbb{N}_{\leq m}$. Each, obstacle $k \in I_o$ is completely determined by its position $x_k \in \mathcal{W}$, and radius $\rho_k \in \mathbb{R}_{>0}$. Thus we will consider that each obstacle $k \in I_o$ occupies an open set $\mathbb{B}(x_k, \rho_k + \rho_\alpha)$.

Note that the subtraction and addition of the agent radius ρ_α in the various definitions, allows us to perceive an agent as a point lying in its center when considering a situation from its point of view, without altering the Euclidean distance of the actual entities.

Remark 2.1. Having defined the notion of obstacles, the only entities involved in the present thesis are either agents or obstacles.

We can now define the *free space* $\mathbb{F}$ as

$$\mathbb{F} \triangleq \mathbb{W} \setminus \bigcup_{k \in I_o} \mathbb{B}(x_k, \rho_k + \rho_\alpha), \tag{2.6}$$

which is essentially the workspace with the obstacles removed, and the free space of each agent $i \in \mathcal{V}$ which is a function of the states of the agents $\mathcal{V} \setminus \{i\}$ as

$$\mathbb{F}_i(\bar{x}_i) \triangleq \mathbb{F} \setminus \bigcup_{j \in \mathcal{V} \setminus \{i\}} \mathbb{B}(x_j, 2\rho_\alpha). \tag{2.7}$$

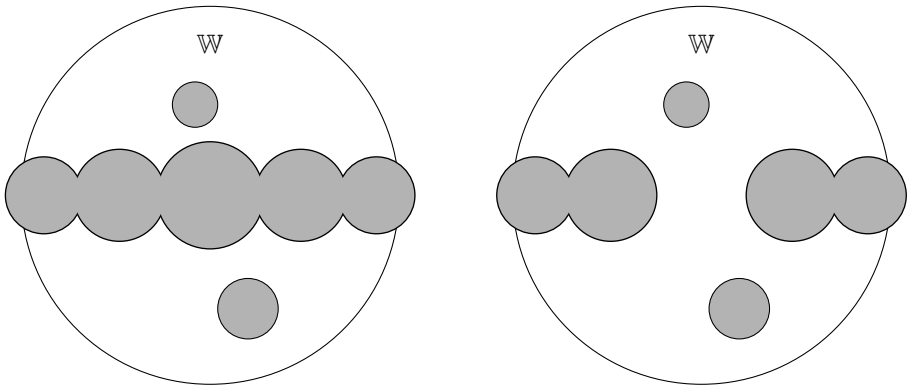


Figure 2.2: Two workspace scenarios. On the left the free space $\mathbb{F}$ is a disconnected set whereas on the right it is connected.

2.2.2 Overlapping Obstacles

The definitions given in the prequel do not prohibit the existence of obstacles that are overlapping. This fact in turns allows for the set $\mathbb{F}$ to become disconnected (see Figure 2.2). This situation can be problematic since it implies that certain tasks on $\mathbb{F}$ are infeasible. Thus we make the following assumption:

Assumption 3 *The free space set $\mathbb{F}$ is connected.*

We make also the following assumption:

Assumption 4 *At the initial time the convex closure of the positions of the agents, $\overline{co}\left(\bigcup_{i\in\mathcal{V}} x(0)_i\right) \subset \mathbb{F}$ and for every t' such that $x_1(t') = x_1^d$, $\overline{co}\left(\bigcup_{i\in\mathcal{V}} x_i(t')\right) \subset \mathbb{F}$ as well.*

In essence Assumption 4 declares that the initial and final configurations are sufficiently away from obstacles.

2.3 Safety Specifications

We now discuss the safety specifications that the system under discussion should respect. As safety specifications we define certain conditions that should not be violated during the operation of the system. For our purposes, the following events should not occur any time t:

1) Collisions,

2) Loss of global connectivity of the communication graph $\mathcal{G}_c$.

Thus, two safety specifications corresponding, one for each of the two events, are defined. Note that the first specification is restricted to a specific portion of the system,i.e., a subset of the agents' states, whereas the second is a global system specification, in the sense that it describes a property which is determined by the full system state x. The preceding notions are now properly developed.

2.4 Collision Avoidance

The collision avoidance specification applies to each agent $i \in \mathcal{V}$ and is thus an *atomic specification*. In particular, every agent must avoid colliding with any other entity, i.e., any obstacle in I_o and every agent in $\mathcal{V} \setminus \{i\}$.

Inter-agent collision avoidance Given a pair of distinct agent $i, j \in \mathcal{V}$, the inter-agent collision avoidance specification is formulated as the following condition:

$$\forall t \geq 0,\ \|x_i(t) - x_j(t)\|_2 > 2\rho_\alpha. \tag{2.8}$$

Note that the condition encoding an inter-agent collision avoidance specification is *symmetric* with respect to the agents involved, due to the positive homogeneity of the norm operator. The construction of controller abiding to this specification is greatly facilitated by the preceding fact.

Obstacle avoidance For an agent $i \in \mathcal{V}$ and an obstacle $k \in I_o$, the obstacle avoidance specification is given by the following inequality:

$$\forall t \geq 0,\ \|x_i(t) - x_k(t)\|_2 > \rho_k + \rho_\alpha. \tag{2.9}$$

Considering (2.8) and (2.9), the collision avoidance specification for our system is summarized as:

$$\forall t \geq 0,\ \forall i \in \mathcal{V},\ x_i \in \mathbb{F}_i(\bar{x}_i). \tag{2.10}$$

2.5 Global 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. Mathematically this is equivalent to requiring that:

$$\forall t \geq 0,\ \mathcal{G}_c\left(x(t)\right) \text{ is connected.} \tag{2.11}$$

2.6 System Objectives

As already mentioned in subsection 2.1, the agents constituting the multi-agent system under investigation are separated into a single *leader* and *followers*. This classification is reflected on the nature of objectives an agent is expected to complete, depending on whether it belongs to one class or the other. Specifically, every agent is expected to maintain a certain formation with respect to some subset of the set of agents.

The leader is burdened with the additional task of navigating from an initial position $x_1(0)$ to a fixed, known desired position in the free space, denoted by x_1^d. These objectives are now developed more extensively.

2.6.1 Formation Specifications

A *formation specification* is essentially a constraint on the states of two distinct agents. Given two agents $i, j \in \mathcal{V}$ and a vector $c_{ij}(t) \in \mathbb{R}^2$, agent i is in formation with agent j at time t if and only if:

$$x_i(t) - x_j(t) = c_{ij}(t) \tag{2.12}$$

The formation objective is formally captured as a *formation graph.*

Definition 2.3 *A formation graph $\mathcal{G}_f(t) = (\mathcal{V}, \mathcal{E}_f(t), \mathcal{C}_f(t))$ is a time-varying graph, where $\mathcal{C}_f(t) = \{c_{ij}(t)\}_{(i,j)\in\mathcal{E}_f(t)}$ is a finite set of formation specifications, indexed by the set $\mathcal{E}_f(t)$. For each $(i, j) \in \mathcal{E}_f(t)$, $c_{ij}(t) \in \mathbb{R}^2$ denotes the desired relative position of agent i with respect to agent j at time t.*

Additionally, we require that the formation specification are not conflicting, in the sense that:

$$\forall (i, j) \in \mathcal{E}_f(t),\ c_{ij}(t) = -c_{ji}(t). \tag{2.13}$$

Again, as in the case of inter-agent collision avoidance, formation specification is symmetric with respect to the agents involved. A very important consequence of this fact, is that agents can successfully attain a formation specification without knowing the position of others with respect to some absolute frame of reference.

The system at state $x(t)$ *models* a formation graph $G_f(t)$, denoted by $x(t) \models \mathcal{G}_f(t)$, if $x_i(t) - x_j(t) = c_{ij}(t)$, $\forall (i, j) \in \mathcal{E}_f(t)$.

Remark 2.2. It is reasonable to require that two agents sharing a formation specification should also *communicate*. Thus, $\mathcal{E}_f(t) \subseteq \mathcal{E}_c(x(t))$, $\forall t \geq 0$ which in turns implies that $\|x_i(t) - x_j(t)\|, \|c_{ij}(t)\| < \rho_c$, $\forall (i, j) \in \mathcal{E}_f(t)$, $\forall t \geq 0$. It then follows that maintaining a connected formation graph guarantees the connectedness of the communication graph as well.

2.6.2 Leader Objectives

Apart from the task of forming a formation with some subset of the set of agents, the leader must in addition navigate from an initial position $x_1(0)$ to a desired position x_1^d in the free space $\mathbb{F}$. Thus, we make the following assumption:

Assumption 5 *It is assumed that there exists a collision-free trajectory from $x_1(0)$ to x_1^d, which the leader knows or can generate.*

In the context of this thesis the aforementioned trajectory will be generated through a Navigation Function [18] controller. Although, certain workspace properties, namely the potential overlap of static obstacles, hinder the construction of an almost globally asymptotically stable controller for the aforementioned task, yet for our purposes pairs of initial and final positions will be chosen so that the initial conditions will not be in the region of attraction of the spurious local minima.

2.6.3 Mathematical Problem Statement

We are now in position to formally state the main problem addressed.

Problem 2.1 *Given an initial connected formation graph $\mathcal{G}_f^0$, with $x(0) \models \mathcal{G}_f^0$ synthesize:*

1) a continuous control input u_i, based solely on local information, for each agent $i \in \mathcal{V}$, and

2) a decentralized controller of the formation graph $\mathcal{G}_f(t)$,

such that no safety specification is violated, and for some time t_f, $x_1(t) = x_1^d$, $\mathcal{G}_f(t) = \mathcal{G}_f^0$ and $x(t) \models \mathcal{G}_f(t), \forall t > t_f$.

Note that in the absence of static obstacles, Problem 2.1 is trivial and the formation graph could be maintained constant and equal to the initial one. However, in the presence of static obstacles reconfiguration of the formation specifications might become necessary in order for the system to complete all of the objectives.

3 Algebraic Topology and Path Homotopy

We now make a digression into algebraic topology, to develop some notions that will prove useful for providing a solution to the problem in hand. The reader who is not interested in the theoretical foundations can skip to *subsection* 3.4.

3.1 Introduction

Consider the motion planning problem for a single robotic agent operating in a cluttered environment. For our purposes the workspace $\mathbb{W}$ is a compact subspace of the two-dimensional Euclidean space and the free space $\mathbb{F}$ is some closed subset of $\mathbb{W}$. Given an initial point and a final point in the free space, denoted by x_0 and x_1, respectively, the motion planning problem consists of finding a path in $\mathbb{F}$ from x_0 to x_1. A path is mathematically defined as:

Definition 3.1 *The continuous map $\gamma : I \to X$ with $\gamma(0) = x_0$ and $\gamma(1) = x_1$ is a path from x_0 to x_1 in X. We say that x_0 is the initial point, and x_1 the final point. The image of γ, $\gamma(I) \subset X$ is a curve in X, where $I \triangleq [0, 1]$.*

Definition 3.2 *A space X is* path-connected *if every pair of points in X can be joined by a path in X.*

Additionally, if γ is a path from x_0 to x_1, there is an *inverse* path, denoted by $\bar{\gamma}$, from x_1 to x_0 defined by the equation:

$$\bar{\gamma}(s) \triangleq \gamma(1 - s), \quad s \in I,$$

and for every point $x \in X$, let ε_x be the constant path at x, i.e.,

$$\varepsilon_x(s) = x. \quad s \in I.$$

Assuming the free space is path-connected[Munkres] a solution to the motion planning problem exists for every pair of initial and final points in $\mathbb{F}$. The solution to every such problem, apart from pathological cases, is however not unique. Consider the collection of paths

$$\Gamma_{\mathbb{F}}(x_0; x_1) \triangleq \{\gamma : I \to \mathbb{F} : \gamma(0) = x_0, \gamma(1) = x_1\}, \tag{3.1}$$

which constitutes the set of solutions of the motion planning problem. The number of elements of $\Gamma_{\mathbb{F}}(x_0; x_1)$ is clearly infinite.

Illustrated in Figure 3.1 is a trivial instance of the motion planning problem involving a single robot and one obstacle, with the free space $\mathbb{F}$ being a compact, connected subset of the Euclidean plane. Six members of the set $\Gamma_{\mathbb{F}}(x_0; x_1)$ are depicted, i.e., six solutions of the motion planning problem. It is immediately evident that a natural way to categorize the solutions into two different categories is in accordance with the adopted coloring. In particular, three solutions, drawn in green, are said to belong to the set $\{\gamma_r\}$, whereas the remaining three, drawn in red, belong to the set $\{\gamma_l\}$. Roughly, the first set is the subset of

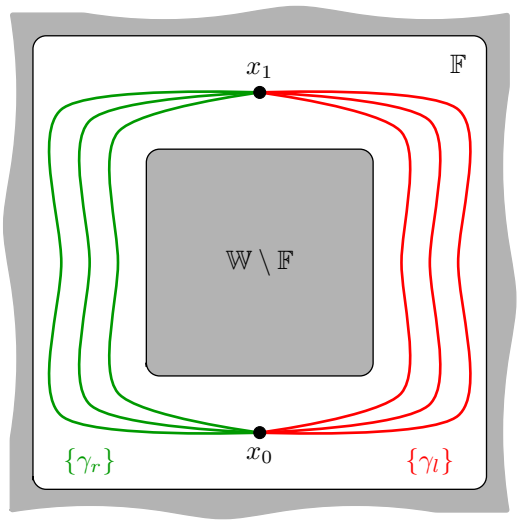


Figure 3.1: A simple instance of the single-robot motion planning problem in the presence of a single obstacle. Three elements of each of the two solution families $\{\gamma_r\}$ and $\{\gamma_l\}$ are depicted.

solutions that "have the obstacle on their right hand side", whereas the latter set is the subset of solutions that "have the obstacle on their left hand side".

We are now going to present the necessary mathematical machinery in order to: (1) Provide a formal definition of the aforementioned rule for categorizing solution paths and (2) Determine the number of different categories that exist for each instance of the motion planning problem.

3.2 Equivalence Relations

What we are essentially trying to achieve is partition the set of paths $\Gamma_{\mathbb{F}}(x_0; x_1)$ into a *finite* number of disjoint sets. In doing so, we borrow some basic notions from Set Theory.

Definition 3.3 *A binary relation on a set X is a subset R of the Cartesian product $X \times X$.*

If R is a relation on X, we use the notation $x \sim_R y$ to denote the fact $(x, y) \in R$. The relation concept is too general for our purpose, thus we limit ourselves to a subset which possesses a number of accommodating properties.

Definition 3.4 *An equivalence relation on a set X is a binary relation R on X having the following three properties:*

1º (Reflexivity) $x \sim_R x$, for every $x \in X$,

2º (Symmetry) $x \sim_R y \Leftrightarrow y \sim_R x$,

3º (Transitivity) If $x \sim_R y$ and $y \sim_R z$, then $x \sim_R z$.

For notational thrift, as long as the relation R is unambiguous we use the "$\sim$" symbol to denote the relation $\sim_R$. Given an equivalence relation $\sim$ on a set X and an element $x \in X$, we can define a subset $[x]$ of X, called the *equivalence class* determined by X, as

$$[x] \triangleq \{z \in X : z \sim x\}. \tag{3.2}$$

Evidently, by the reflexivity property of $\sim$, $x \in [x]$. Additionally, equivalence classes possess the following important property:

Lemma 3.1 *Two equivalent classes $[x]$, $[y]$ are either disjoint or equal.*

Proof. Please refer to proof of Lemma 3.1 in [19]. □

Since every element of A belongs to some equivalence class, induced by an equivalence relation, and different classes are disjoint, it follows that equivalence classes constitute a partition of the set A into disjoint sets.

The set of all equivalence classes in X induced by the equivalence relation $\sim$ is denoted as $X/\sim$ and is called the quotient set of X by $\sim$. A simple example is given to clarify the preceding concept:

Example 3.1. Consider the set $A = \mathbb{R} \setminus \{0\}$ and take the equivalence relation $\sim = \{(x, y) \in A \times A : xy > 0\}$. Under this equivalence relation two equivalence classes are induced, namely $[x_{>0}] = \{x \in A : x > 0\}$ and $[x_{<0}] = \{x \in A : x < 0\}$, that is the classes of positive and negative real numbers, respectively. It follows that the quotient set has two elements, $X/\sim = \{[x_{>0}], [x_{<0}]\}$ and consequently the set A can be written as the union of two disjoints sets, i.e., $A = (-\infty, 0) \cup (0, \infty)$.

Note that equivalence classes allow us to partition sets which are infinite, and even uncountable as in Example 3.1, into a number of finite disjoint sets. We are now ready to expand this concept into sets of paths in order to perform a similar partitioning.

3.3 Path Homotopy

The intuitive concept illustrated in Figure 3.1 that allowed use to categorize the paths that constituted solutions to the motion planning problem can be mathematically formalized using the notion of *path homotopy*.

Definition 3.5 *Two paths γ, γ', mapping the interval I into X are said to be path homotopic if they have the same initial point x_0 and final point x_1, and if there exists a continuous map $H : I \times I \to X$ such that*

$$H(s, 0) = \gamma(s) \quad \text{and} \quad H(s, 1) = \gamma'(s), \tag{p1}$$
$$H(0, t) = x_0 \quad \text{and} \quad H(1, t) = x_1, \tag{p2}$$

for each $s \in I$ and each $t \in I$. We call H a path homotopy between γ and γ'. If γ is path homotopic to γ', we write $f \simeq_p g$.

The first property (p1) suggests that H is a continuous way of "deforming" path γ into γ', whereas the second property (p2) says that the endpoints of the path remain fixed during deformation.

Proposition 3.1 *The relation $\simeq_p$ is an equivalence relation.*

Proof. We verify that the three properties of an equivalence relation, namely reflexivity, symmetry and transitivity, hold.

Proving reflexivity is trivial, i.e., $\gamma \simeq_p \gamma$, since $H(s,t) = \gamma(s)$ is a valid path homotopy for that case.

To prove symmetry, i.e., $\gamma \simeq_p \gamma' \Leftrightarrow \gamma' \simeq_p \gamma$, consider a path-homotopy H between γ and γ', then $H' = H(s,(1-t))$ is a path-homotopy between γ' and γ.

Finally, given $\gamma_1 \simeq_p \gamma_2$, $\gamma_2 \simeq_p \gamma_3$ and corresponding path homotopies H_1, H_2, we define $H_3 : I \times I \to X$ as

$$H_3(s,t) = \begin{cases} H_1(s,2t), & t \in \left[0, \frac{1}{2}\right] \\ H_2(s,2t-1), & t \in \left[\frac{1}{2}, 1\right] \end{cases}. \tag{3.3}$$

The map H_3 is well defined, since for $t = \frac{1}{2}$, $H_1(s,2t) = g(s) = H_2(s,2t-1)$. H_3 is continuous on the two closed subsets $I \times [0, \frac{1}{2}]$ and $I \times [\frac{1}{2}, 1]$, and thus by the pasting lemma, it is also continuous on all of $I \times I$. Consequently, H_3 is the required path homotopy between H_1 and H_2 and transitivity is verified. □

Since path homotopy is an equivalence relation, the set of paths in a space X can be partitioned into equivalence classes. Thus, given a path γ, we denote the class of paths that are homotopic equivalent to γ by $[\gamma]$.

Example 3.2. Let $\gamma_1, \gamma_2 : I \to X$ be two paths in in $X = \mathbb{R}^2$ with the same endpoints x_0 and x_1. It is easy to see that γ_1 and γ_2 are path-homotopic; the map

$$H_l(s,t) \triangleq (1-t)\gamma_1(s) + t\gamma_2(s) \tag{3.4}$$

is a path homotopy between them called the *straight-line homotopy* since it moves every point $\gamma_1(s)$ to the corresponding point $\gamma_2(s)$ along the straight-line segment joining them. In Figure 3.2, the two paths, γ_1 and γ_2, are illustrated as well as the straight-line path homotopies at three points. The red path is the straight-line homotopic path for $t = \frac{1}{2}$.

More generally, for any *convex* subspace A of $\mathbb{R}^n$, any two paths γ, γ' in A from x_0 to x_1 are path homotopic in A, for the straight-line homotopy H_l between them has image contained in A, as a direct consequence of the convexity of A.

Example 3.3. Let $X = \mathbb{R}^2 \setminus \{(0,0)\}$, i.e., the *punctured plane*. The following paths in X,

$$\gamma_1(s) = (\cos(\pi s), \sin(\pi s)),$$
$$\gamma_2(s) = (\cos(\pi s), 1.5\sin(\pi s)),$$

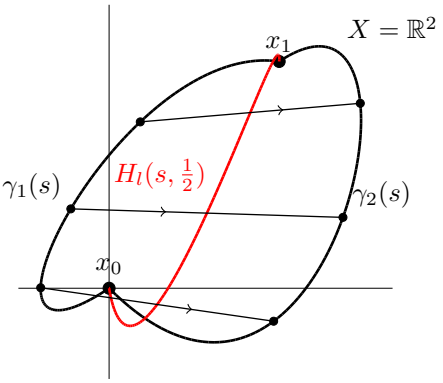


Figure 3.2: Two straight-line homotopic paths γ_1 and γ_2.

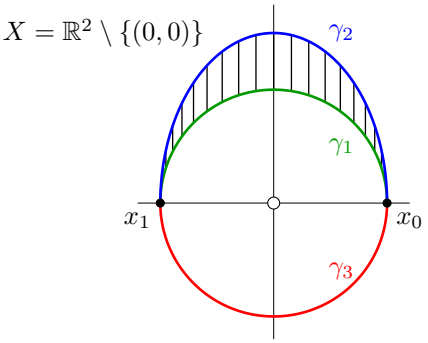


Figure 3.3

are path homotopic, since the straight-line homotopy between them is an acceptable path homotopy. However, the straight-line homotopy between γ_1 and the path

$$\gamma_3(s) = (\cos(\pi s), -\sin(\pi s)),$$

is not acceptable for its image is not a subset of X (see Figure 3.3).

It turns out that there exists no path homotopy in X between paths γ_1 and γ_3. It is intuitively quite evident that "deforming" γ_1 into γ_3 without passing through the point $(0,0)$ is impossible without introducing a discontinuity. The formal proof is however more involved.

An important observation extracted from the preceding example is that one cannot assert whether two paths are path homotopic without knowing the codomain set of the paths, X. In Example 3.3, paths γ_1 and γ_3 would be path homotopic if $X = \mathbb{R}^2$.

We now introduce an *operation* on paths called the *product*.

Definition 3.6 *If γ_1 is a path in X from x_0 to x_1, and if γ_2 is a path in X from x_1 to x_2, we define the* ***product*** *$\gamma_1 \cdot \gamma_2$ of γ_1 and γ_2 to be the path γ_3 given by the equation*

$$\gamma_3(s) \triangleq \begin{cases} \gamma_1(2s), & s \in [0, \frac{1}{2}] \\ \gamma_2(2s-1), & s \in [\frac{1}{2}, 1] \end{cases}. \tag{3.5}$$

The function γ_3 is well-defined and continuous, by the pasting lemma, and defines a path in X from x_0 to x_2.

It is worth noting that the path product operation on paths induces a well-defined operation on path-homotopy classes, defined by the equation

$$[\gamma_1] \cdot [\gamma_2] \triangleq [\gamma_1 \cdot \gamma_2]. \tag{3.6}$$

The class of paths $[\gamma_1 \cdot \gamma_2]$ is determined by the path homotopy defined as

$$H_3(s,t) = \begin{cases} H_1(2s,t), & s \in [0, \frac{1}{2}] \\ H_2(2s-1,t), & s \in [\frac{1}{2}, 1] \end{cases}, \tag{3.7}$$

where H_1, H_2 are path homotopies of the $[\gamma_1]$ and $[\gamma_2]$ classes, respectively. One can regard $[\gamma_1 \cdot \gamma_2]$ as the equivalence class of all paths in X that start at x_0, pass through x_1 and end at x_2.

Theorem 3.1 (Properties of product operation on paths)

At this point, it is important to remind that the product operation on path-homotopy classes is only defined for pairs of path homotopy classes $[\gamma_1], [\gamma_2]$ satisfying $\gamma_1(1) = \gamma_2(0)$.

3.3.1 The Fundamental Group

The fact that the product operation (3.6), is only defined for pairs of paths γ_1, γ_2 with $\gamma_1(1) = \gamma_2(0)$, leads us to restrict are attention to paths that begin and end at the same point. In this, setting the product operation can always be well-defined. Such paths are called *loops*.

Definition 3.7 *Let X be a space. A* loop *in X based at x_0 is a path $l : I \to X$ such that $l(0) = l(1) = x_0$.*

It turns out [20] that the set of path homotopy classes of loops based at some point $x_0 \in X$ forms a group under the path product operation. This group is called the *fundamental group* of X.

Definition 3.8 *A* group *is a set G together with an operation $\cdot$ that combines any two elements a and b to form another element, denoted by $a \cdot b$. To qualify as a group, the set and operation $(G, \cdot)$, must satisfy four requirements known as the group axioms:*

1º (Closure) For all $a, b \in G$, $a \cdot b \in G$,

2^o *(Associativity) For all* $a, b, c \in G$, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$,

3^o *(Identity element) There exists an element* e*, such that for every element* $a \in G$*, the equation* $a \cdot e = e \cdot a = a$ *holds. Such an element is* unique *and is called the identity element.*

4^o *(Inverse element) For each* $a \in G$*, there exists an element* $b \in G$*, denoted by* a^{-1}*, such that* $a \cdot b = b \cdot a = e$*, where* e *is the identity element.*

A *trivial group* is a group consisting of a single element which by definition is the *identity element*, e.

Definition 3.9 *The* fundamental group *of a space* X *relative to the base point* x_0*, denoted by* $\pi_1(X, x_0)$*, is the set of path homotopy classes of loops based at* x_0*.*

If follows from Theorem 3.1, that the product operation, when restricted to the subset of path homotopy classes of loops based at some point x_0, satisfies the group axioms. Given two loops l_1, l_2 in X, based at x_0, the product $l_1 \cdot l_2$ is always defined and is a *loop* based at x_0, validating the *closure* axiom. Note that the *identity element* of the fundamental group $\pi_1(X, x_0)$ is $[\varepsilon_{x_0}]$ and the *inverse element* of $[l]$ is $[\bar{l}]$.

Example 3.4. Let X be a *convex* subset of $\mathbb{R}^n$, and l be a loop based at x_0. Then the straight-line homotopy is a path homotopy between l and the constant path at x_0. Thus, $l \simeq_p \varepsilon_{x_0}$ and $[l] = [\varepsilon_{x_0}]$, i.e., the fundamental group $\pi_1(X, x_0)$ is the trivial group.

Another interesting question is to what extend does the fundamental group depend on the choice of the base point.

Theorem 3.2 (Change of Base Point) *Suppose* X *is path-connected,* $x_0, x_1 \in X$ *and* γ *is a path from* x_0 *to* x_1*. The map* $\Phi_\gamma : \pi_1(X, x_0) \to \pi_1(X, x_1)$ *defined by*

$$\Phi_\gamma([l]) \triangleq [\bar{\gamma}] \cdot [l] \cdot [\gamma],$$

is an isomorphism.

Proof. The mapping Φ_g is well defined since the corresponding product operations are always well-defined. We now show that Φ_g is a group homomorphism:

$$\begin{aligned} \Phi_\gamma([l_1]) \cdot \Phi_\gamma([l_1]) = & \\ &= [\bar{\gamma}] \cdot [l_1] \cdot [\gamma] \cdot [\bar{\gamma}] \cdot [l_2] \cdot [\gamma] \\ &= [\bar{\gamma}] \cdot [l_1] \cdot [l_2] \cdot [\gamma] \\ &= \Phi_\gamma([l_1] \cdot [l_2]). \end{aligned}$$

Being a homomorphism, Φ_g is also an isomorphism, since it has an inverse given by $\Phi_{g^{-1}} : \pi_1(X, x_1) \to \pi_1(X, x_0)$. □

Corollary 3.1 *If X is path-connected and $x_0, x_1 \in X$, then $\pi_1(X, x_0)$ is* isomorphic *to $\pi_1(X, x_1)$.*

Although the fundamental groups of a path-connected space X based at any two points are isomorphic, different paths between two points may give rise to different isomorphisms between the corresponding fundamental groups. In particular, the isomorphism of $\pi_1(X, x_0)$ with $\pi_1(X, x_1)$ is independent of the path if and only if the fundamental group is *abelian*, i.e., the group operation *commutes*.

Definition 3.10 *A space X is said to be* simply connected *if it is a path-connected space and if $\pi_1(X, x_0)$ is the trivial group for some x_0, and hence, by Corollary 3.1, for every $x_0 \in X$.*

The following lemma follows:

Lemma 3.2 *In a simply connected space X, any two paths having the same initial and final points are path homotopic.*

Proof. Let γ_1, γ_2 be two paths in X from x_0 to x_1. Then the product $\gamma_1 \cdot \gamma_2^{-1}$ is defined and is a loop on X based at x_0. Since X is simply connected, this loop is path homotopic to the constant loop at x_0, i.e., $[\gamma_1 \cdot \gamma_2^{-1}] = [\varepsilon_{x_0}]$. Then,

$$\begin{aligned}
&[\gamma_1 \cdot \gamma_2^{-1}] \cdot \gamma_2 = [\varepsilon_{x_0}] \cdot [\gamma_2] && \overset{(3.6)}{\Longleftrightarrow} \\
&[\gamma_1] \cdot [\gamma_2^{-1} \cdot \gamma_2] = \gamma_2 && \Longleftrightarrow \\
&[\gamma_1] \cdot [\varepsilon_{x_0}] = \gamma_2 && \Longleftrightarrow \\
&[\gamma_1] = [\gamma_2], &&
\end{aligned}$$

completing the proof. □

3.4 Multi-agent Path Homotopy

In the problem under investigation, the leader's role is to implicitly guide the system towards the desired configuration. The followers, attempting to maintain their formation specifications follow along. However, the very fact that obstacles are not disjoint can lead to the entrapment of a number of followers to some subset of the free space from where no further progress of the system is possible unless some sort of backtracking is performed. It is obvious that this kind of behaviour is problematic and should be avoided. The higher the number of agents the more difficult the accomplishment of the aforementioned task becomes.

In these lines, we will utilize the notion of path homotopy to ensure that all agents negotiate every obstacle from the "same side". Note that since all agents have the same radius, the fact that the leader can get to its desired point implies that this is also the case for the followers.

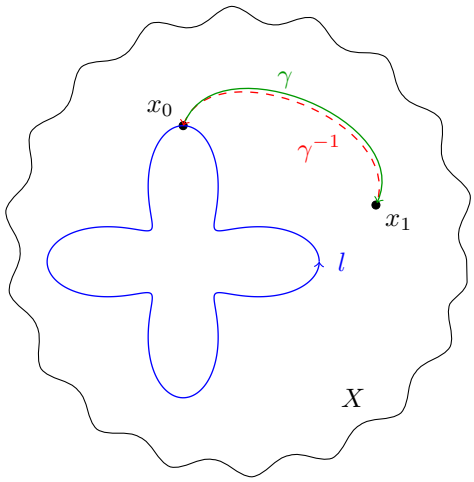


Figure 3.4

The trajectories of different agents, define paths that do not have the same initial and final points. Thus, applying the notion of path homotopy as already defined is not possible.

To overcome this difficulty, we take a slightly different approach. We impose an additional restriction to be fulfilled by the trajectory of the system. First, let us define the set

$$\mathbb{F}_h \triangleq \mathbb{W} \setminus \bigcup_{k \in I_o} x_k \tag{3.8}$$

i.e, the workspace with the centers of the obstacles removed. In this respect, we require that:

$$\overline{co}(\{x_i(t), x_j(t)\}) \subset \mathbb{F}_h, \, \forall (i,j,k) \in \mathcal{E}_f(t) \times I_o, \, \forall t \geq 0, \tag{3.9}$$

where $\overline{co}(\cdot)$ denotes the convex closure operator.

which is equivalent to:

$$\overbrace{\frac{x_i(t) - x_k}{\|x_i(t) - x_k\|}}^{\hat{x}_{ik}} \cdot \overbrace{\frac{x_j(t) - x_k}{\|x_j(t) - x_k\|}}^{\hat{x}_{jk}} + 1 > 0, \forall (i,j,k) \in \mathcal{E}_f(t) \times I_o, \forall t \geq 0 \tag{3.10}$$

where $|\mathcal{E}_f(t)|$ is a non-increasing function of time (see Figure 3.5).

This property also implies that the set $\mathbb{F} \cap \bigcup_{i \in \mathcal{V}} \mathbb{B}(x_i, \frac{\rho_c}{2})$ is path connected at all times. Additionally, for each agent there exists a feasible direction of movement that decreases its distance with respect to some of its formation neighbors. If we allow $|\mathcal{E}_f(t)|$ to increase this might not hold.

We will refer to this property as *path homotopy*. A system trajectory that satisfies (3.10) possesses the property that all agents will eventually overcome every obstacle from the same "side" as the leader.

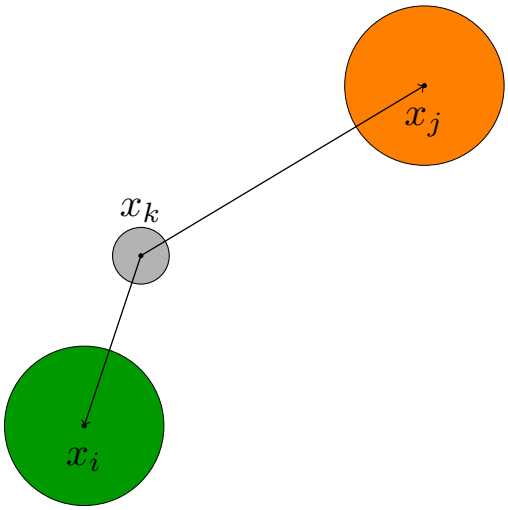


Figure 3.5: Illustration of the definition of (3.10). If x_i, x_j, x_k are aligned, path homotopy is violated.

This property is of exceptional importance, in the present setting, since the fact that the obstacles are not disjoint may lead to some number of followers getting "trapped" to a subset of the workspace, that would require the leader to backtrack for the system to be able to reach the desired configuration. Even though by the Assumption 5, the leader can always find its way around the obstacles towards its desired position, this is not in general true for the followers. Thus, imposing path homotopy constitutes an implicit method of propagating this information to the followers. Finally note that by Assumption 4 the initial condition of the system, $x(0)$, satisfies (3.10).

4 Decentralized Navigation Functions-based Controller

4.1 Introduction

The Navigation Functions method introduced in the seminal work of Koditschek and Rimon [18] enabled the guaranteed convergence of a single agent in an environment with disjoint disk shaped obstacles. In [21] the results were extended to the case of multiple agents. In the present thesis, we adopt a similar formulation for the problem in hand. We prove certain properties of the controller indicating that it is partially suitable for this particular problem. Yet, certain limitations, due to the coexistence of static obstacles and connectivity maintenance appear, rendering the task of proving (almost) global asymptotic stability impossible. Nevertheless, the proposed controller can be used as a low-level controller in a multi-layered control scheme.

4.2 Potential Functions Design

Consider a DNFs based potential $\phi_i : \mathbb{F}_i \to [0, 1]$

$$\varphi_i \triangleq \frac{\gamma_i}{(\gamma_i^{\kappa} + \beta_i)^{\frac{1}{\kappa}}}, \quad i \in \mathcal{V}, \tag{4.1}$$

where $\kappa \in \mathbb{R}_{>0}$ is a gain parameter, while $\gamma_i : \mathbb{R}^2 \to \mathbb{R}$ and $\beta_i : \mathbb{R}^2 \to \mathbb{R}$ are functions that encode the desired behaviour[1] and safety/path homotopy specifications of agent i, respectively. We now explicitly define each term in (4.2).

4.2.1 Encoding Objectives

For each agent $i \in \mathcal{V}$, the γ_i term in (4.2) encodes the agent's objectives. In particular,

$$\gamma_i \triangleq \begin{cases} \left\|x_i - x_i^d\right\|^2 + \sum\limits_{j \in \mathcal{N}_i(\mathcal{G}_f)} \left\|x_i - x_j - c_{ij}\right\|^2, & i = 1 \\ \sum\limits_{j \in \mathcal{N}_i(\mathcal{G}_f)} \left\|x_i - x_j - c_{ij}\right\|^2, & i \neq 1 \end{cases} .s \tag{4.2}$$

For all agents, γ_i involves a sum of terms corresponding to their respective formation specifications. The leader has an additional term that corresponds to the desired final position it must attain.

[1] The formation graph $\mathcal{G}_f$ is considered piecewise constant.

4.2.2 Safety Maintenance

Safety conditions for each individual agent are encoded into the β_i term which is defined as:

$$\beta_i \triangleq B_i^{\mathbb{W}} \prod_{k \in \mathcal{N}_i(\mathcal{G}_c), k \in I_o} B_{ik} \prod_{j \in \mathcal{N}_i(\mathcal{G}_f)} b_{ij} \prod_{(j,k) \in \mathcal{N}_i(\mathcal{G}_f) \times I_o} b_{ijk}. \tag{4.3}$$

The first term $B_i^{\mathbb{W}} \triangleq \sigma_{\epsilon_1}^1(\rho_w - \|x_i\|_2)$ is responsible for retaining the agent inside the workspace, whereas the terms $B_{ik} \triangleq \sigma_{\epsilon_2}^1(\|x_i - x_k\|_2 - (\rho_k + \rho_i))$ are responsible for collision avoidance with other agents and static obstacles. Additionally, connectivity and path homotopy maintenance is enforced by $b_{ij} \triangleq \sigma_{\epsilon_3,R}^2(\|x_i - x_j\|_2)$ and $b_{ijk} \triangleq \sigma_{\epsilon_4}^1(1 + \hat{x}_{ik} \cdot \hat{x}_{jk})$ respectively.

The functions $\sigma_\epsilon^1 : \mathbb{R} \to \mathbb{R}_{\geq 0}$ and $\sigma_{\epsilon,R}^2 : \mathbb{R} \to [0,1]$ are switches that are used to normalize the constraint functions to the corresponding codomains and preserve smoothness in the context of limited sensing, by providing a buffer region determined by the positive parameter ϵ. The aforementioned functions are defined by:

$$\sigma_\epsilon^1(z) \triangleq \begin{cases} -\frac{1}{\epsilon^2}z^2 + \frac{2}{\epsilon}z, & z < \epsilon \\ 1, & z \geq \epsilon \end{cases}, \tag{4.4}$$

and

$$\sigma_{\epsilon,R}^2(z) \triangleq \begin{cases} 1, & z \leq R - \epsilon \\ -\frac{1}{\epsilon^2}(z + 2\epsilon - R)^2 + \frac{2}{\epsilon}(z + 2\epsilon - R), & R - \epsilon < z < R \\ 0, & z \geq R \end{cases}, \tag{4.5}$$

respectively.

The graphs of both switches are depicted in Figure 4.1. In both cases, the parameter ϵ controls the width of the buffer, i.e., how quickly the switch changes its value from 0 to 1. In the case of $\sigma_{\epsilon,R}^2$, the positive parameter R is actually equal to $\inf(\sigma_{\epsilon,R}^2)^{-1}(\{0\})$, i.e., the smallest value for which the switch attains a zero value.

4.3 Controller

We now define a decentralized controller for each agent $i \in \mathcal{V}$ which is derived from the negated gradient of the corresponding potential function. We then prove certain properties of this controller.

4.3.1 Definition

The control input for each agent is given by:

$$u_i(t) = -\nabla_i \varphi_i, \quad i \in \mathcal{V}, \tag{4.6}$$

where $\nabla_i \varphi_i$ denotes the gradient of φ_i with respect to x_i.

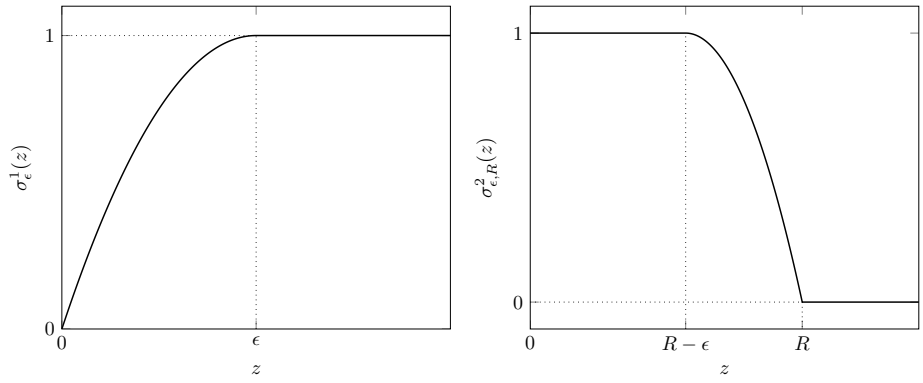


Figure 4.1: The graphs of the σ_ϵ^1 and $\sigma_{\epsilon,R}^2$ switches, on the left and right, respectively.

Remark 4.1. The above defined controller is decentralized in the sense that, the calculation of the control input u_i at each time instant t depends only on values that are readily available to agent i, either through *measurement* or *direct communication*.

4.3.2 Properties

The controller defined in (4.6) possesses certain properties which we now state and prove.

Safety It is of paramount importance that every controller guarantees certain safety specifications. In our case, those have been implicitly encoded through the β_i terms of the potential functions in (4.2). Safety specifications as well as the path homotopy property have have been encoded as (virtual) obstacles, with their boundaries being the pre-images of the zero levels set of the β_i functions, i.e., $\beta^{-1}(\{0\})$. Then, the establishment of the safety properties for controller (4.6) reduces to the following proposition:

Proposition 4.1 *The set* $\{x \in \mathbb{R}^{2n} : \beta_i > 0, \forall i \in \mathcal{V}\}$ *is invariant for the trajectories of* (2.1) *under the control law* (4.6).

Proof. The proof is similar to that of Lemma 2 in [14]. □

Proposition 4.1 establishes the maintenance of the safety specifications and the path homotopy property.

Convergence Although the velocity of each agent is the negated gradient of its corresponding potential ϕ_i, the overall system is not a gradient system [22], as in the case of single-agents Navigation Functions. Thus convergence to the

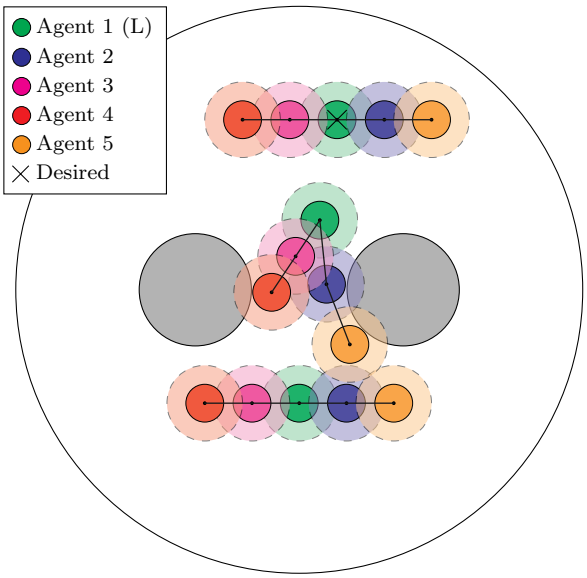


Figure 4.2: A case with $n = 5$ agents where the DNFs based controller successfully stabilizes the system to the desired configuration. The configuration of the system at 3 time instants is depicted.

set $\{x \in \mathbb{R}^{2n} : \|\nabla_i \varphi_i\|_2 = 0\}$, i.e., the set of equilibria is not a priori guaranteed. In fact, we prove the following weaker proposition:

Proposition 4.2 *Given any $\varepsilon \in \mathbb{R}_{>0}$, there exists $\bar{\kappa}(\varepsilon) \in \mathbb{R}_{>0}$ such that for $\kappa > \bar{\kappa}$ system (2.1) under the control law (4.6) converges to the set $\mathcal{C}_\phi \triangleq \{x \in \mathbb{R}^{2n} : \|\nabla_i \varphi_i\| \leq \varepsilon, \forall i \in \mathcal{V}\}$.*

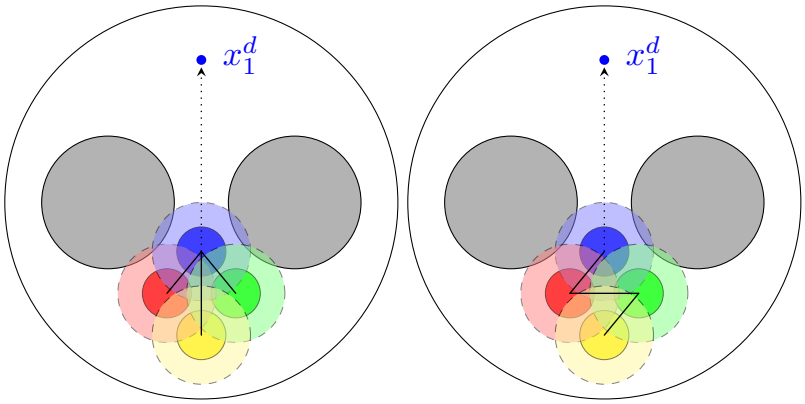
Proof. For the proof please refer to [23]. □

Remark 4.2. The condition of Proposition 4.2 is *sufficient*. In practice, convergence to some equilibrium occurs irrespectively of the value of κ.

By Proposition 4.2, convergence to some arbitrarily small neighborhood of the set of critical points is established for a sufficiently large value of the parameter κ. Note that this is a *sufficient* condition, meaning that in practice It is easy to see that the desired configuration belongs to the set $\mathcal{C}_\varphi$ since $\gamma_i = \nabla_i \gamma_i = 0, \forall i \in \mathcal{V}$, implying that $u_i = 0, \forall i \in \mathcal{V}$. However, despite the previous fact, ensuring that the system will converge to this point is not straightforward. The reasons for this fact are now analysed.

4.4 The Need for Reconfiguration

The DNFs based controller proposed in this section can in some cases stabilize the system to the desired configuration. However, there exist cases where this



(a) Infeasible task. (b) Feasible task.

Figure 4.3: A case illustrating the need for reconfiguration of the set of formation/connectivity specifications.

cannot be achieved seldom by using the aforementioned controller. The reasons for inadequacy are twofold.

On the one hand, the existence of static obstacles in conjunction with connectivity maintenance specifications can render the task of navigating the system from the initial to the desired configuration infeasible. On the other hand, even if the later is not the case, it turns that, for certain instances of the problem under investigation, the appearance of unwanted local minima cannot be avoided. Thus, (almost) global asymptotic stability cannot be claimed.

4.4.1 Task Infeasibility

As already mentioned, the coexistence of static obstacles and connectivity maintenance specifications can render the task of navigating from an initial to a desired configuration infeasible. In such cases, reconfiguration of the set of formation, and consequently connectivity, specifications is necessary for the initially the leader, and eventually the followers, to be able to progress towards the desired configuration.

As an example, consider the case illustrated in Figure 4.3. Given a formation graph as the one depicted in 4.3a, it is impossible for the system to reach the desired configuration, without either a collision or a violation of some connectivity maintenance occurring. Then, since the desired configuration does not belong to the forward invariant set of system (2.1) under (4.6), by the Extreme Value Theorem, the restriction of every ϕ_i in the closure of that invariant set must attain some other minimum value, thus establishing the existence of some other configuration with a dense region of attraction. However, for a different formation graph and consequently different connectivity constraints, this is no longer the case (see 4.3b).

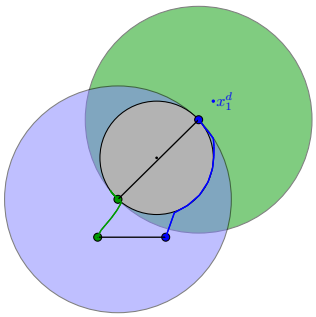


Figure 4.4: A simple example with two agents and one obstacle illustrating the persistence of an unwanted local minimum. The diameter of the obstacle equals the communication radius. The opaque disks represent the connectivity region for each agent.

4.4.2 Persistence of Local Minima

On the other hand, proving convergence is still elusive, even though it might not be impossible for the system to reach the desired configuration. Notice that the existence of unwanted critical points is a topological inevitability [24]; thus the best one can hope for is global asymptotic stability except for a set of initial conditions of zero measure. Proving that would require, some upper bound on the value of the parameter κ that would ensure that unwanted critical points are saddles. However, it turns out that such a bound may not exist even for some simple examples.

For instance, considering the almost trivial case illustrated in Figure 4.4, it can be easily noticed that as the value of κ increases, the agents will negotiate the obstacle from opposite sides. Hence, as they reach diametrically opposite positions with respect to the obstacle, both connectivity and collision constraints tend to be violated simultaneously for both agents, i.e., the boundary of the obstacle tends to coincide with the boundary of the connectivity region. Consequently, a local minimum appears, as there exists no direction of movement for the system that can further reduce either φ_i. The system actually converges to the same local minimum for a continuum of initial conditions.

4.5 Discussion

The controller proposed in this section can solve certain instances of the problem in hand. However, as we've seen in the previous subsection this is not always the case. Thus, the construction of a reconfiguration algorithm is deemed necessary, in order to address the whole range of problems. Additionally, it will prove useful to develop a control algorithm that guarantees convergence for some subclass of problems, such as the one illustrated in Figure 4.4. Those steps will lead towards a provably correct integrated algorithm.

Contents

5 Reconfiguration Strategy

As we have seen in the prequel, the DNFs based controller cannot always successfully stabilize the system in the desire configuration. In those cases, a *distributed discrete algorithm* is employed aiming at reconfiguring the set of formation specifications so that progress towards the desired configuration can be made. This is achieved through the exploitation of information generated during the solving process of an appropriately formulated Distributed Constraint Satisfaction Problem.

5.1 Constraint Satisfaction Problem

5.1.1 Preliminaries

A *Constraint Satisfaction Problem (CSP)* is a problem involving a finite set of *variables*, a finite set of *domains* and a finite set of *constraints*. Variables are allowed to take values that belong to their respective domains and each constraint restricts the combination of values that the set of variables it involves can attain.

Definition 5.1 *A* Constraint Satisfaction Problem *is defined by a* 3*-tuple* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$*, where:*

- $\mathcal{Q} = \{q_1, q_2, \ldots, q_n\}$ *is a set of* n variables*;*
- $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ *is a set of* n domains*, where* $D(x_i)$ *is a finite set of possible values variable* x_i *may attain;*
- $\mathcal{C} = \{C_1, C_2, \ldots, C_e\}$ *is a set of* e constraints *that describe the combinations of values allowed for the variables they involve. The variables involved in a constraint* $C_i \in \mathcal{C}$ *are a called its scope(*$\texttt{scope}(q_i) \subseteq \mathcal{Q}$*).*

Mathematically, every constraint is a subset of the Cartesian product of the domains of its scope, i.e., $C_i \subseteq \prod_{k \in \texttt{scope}(C_i)} D_k$. The number of elements of the scope of a constraint $C_i \in \mathcal{C}$ is called the arity of the constraint C_i.

The class of constraints involving two variables is called *binary*. For our purposes, binary constraints are sufficient, nonetheless a constraint of any arity can be expressed as a set of binary constraints [25]. We will denote a binary constraint in $\mathcal{C}$ between variables q_i and q_j by c_{ij}, with $c_{ij} \subseteq D_i \times D_j$. A CSP where all constraints are binary is called a *Binary Constraint Satisfaction Problem*, and can be represented by a *graph*.

Proposition 5.1 *A* Binary Constraint Satisfaction Problem *can be represented by a* constraint graph $\mathcal{G}_\mathcal{C} = (\mathcal{V}(\mathcal{G}_\mathcal{C}), \mathcal{E}(\mathcal{G}_\mathcal{C}))$*, where the vertices correspond to the variables of the problem* $(\mathcal{V}(\mathcal{G}_\mathcal{C}) = \mathcal{X})$ *and the edges* $\mathcal{E}(\mathcal{G}_\mathcal{C})$ *represent the constraints, i.e.,* $(q_i, q_j) \in \mathcal{E}(\mathcal{G}_\mathcal{C}) \Leftrightarrow C_{ij} \in \mathcal{C}$*.*

The fact that a CSP has an equivalent representation as a graph allows us to borrow notions from graph theory and use them in the context of the original problem.

Definition 5.2 *Two variables are* adjacent *iff they share a constraint, i.e.,* $C_{ij} \in \mathcal{C}$. *If* q_i *and* q_j *are adjacent we say that* q_i *and* q_j *are* neighbors. *The set of neighbors of a variable* q_i*, denoted by* $\Gamma(q_i)$*, is defined as*

$$\Gamma(q_i) \triangleq \{q_j \in \mathcal{X} : C_{ij} \in C\}.$$

Solving a CSP is equivalent to finding a combination of assignments of values to each variable such that all constraints are satisfied.

We now present some examples of problems that can be formulated as constraint satisfaction problems.

5.1.2 Example of CSP

The n-queens problem is a well known combinatorial problem that can be formulated as a CSP. The n-queens problem is essentially the problem of placing n queens on an $n \times n$ chessboard in such a way that no queen can capture another queen. Specifically, two queens can capture each other if they are located on the same row, column or diagonal on the chessboard. As an example, let us formulate the 4-queens problem as a binary CSP:

- $\mathcal{Q} = \{q_1, q_2, q_3, q_4\}$, where each variable q_i is the row placement of each queen that corresponds to every row.
- $D = \{D_1, D_2, D_3, D_4\}$, where $D_i = \{1, 2, 3, 4\}$, $\forall i \in \{1, 2, 3, 4\}$. The elements of D_i are essentially the different possible rows that the queen corresponding to each column can be placed.
- $\mathcal{C} = \{C_{ij} : (q_i \neq q_j) \wedge (|q_i - q_j| \neq |i - j|), \forall i, j \in \{1, 2, 3, 4\}, i \neq j\}$

5.2 Distributed Constraint Satisfaction Problem

5.2.1 Preliminaries

A *Distributed Constraint Satisfaction Problem (DisCSP)* is a CSP where variables and constraints are distributed among multiple automated agents [26].

Definition 5.3 *A* Distributed Constraint Satisfaction Problem *is defined by a* 4*-tuple* $(\mathcal{V}, \mathcal{X}, \mathcal{D}, \mathcal{C})$*, where:*

- $\mathcal{V} = \{A_1, A_2, \ldots, A_n\}$ *is a set of* m agents*;*
- $\mathcal{Q} = \{q_1, q_2, \ldots, q_n\}$ *is a set of* n variables*, with each variable controlled by a single agent in* $\mathcal{A}$*;*
- $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ *is a set of* n domains*, where* $D(x_i)$ *is a finite set of possible values variable* x_i *may attain;*

- $\mathcal{C} = \{C_1, C_2, \ldots, C_e\}$ *is a set of* e constraints *that describe the combinations of values allowed for the variables they involve.*

In the present work, we shall make the following assumptions regarding the nature of the problem:

(i) Each agent controls a *single* variable, i.e., $n = m$.

(ii) Constraints are *binary*.

(iii) Each agent knows every constraint associated with its variable and consequently all of its *neighbors*.

In the present thesis and in the context of a DisCSP agents communicate under the model proposed in [27] where it is assumed that:

(i) Agents communicate by exchanging messages;

(ii) The delay in delivering a message is random but finite.

Agents' Ordering We are now going to equip the set of agents with a *strict total order* [28] induced by the relation $\prec$. We assume that this strict total order is the *lexicographic order* of the set of agents, $[A_1, A_2, \ldots, A_n]$. Then,

$$(\forall i)(\forall j \neq i)\ (A_i \prec A_j \Leftrightarrow i < j).$$

Then for each agent $A_i \in \mathcal{A}$, an agent $A_j \in \mathcal{A}$ has a *higher priority* than A_i iff $A_j \prec A_i$. Conversely, A_j has *lower priority* than A_i iff $A_i \prec A_j$. Using this relation, the set of neighbors of an agent $A_i \in \mathcal{A}$, $\Gamma(A_i)$, can be partitioned in two disjoint sets, the set of *higher-priority neighbors*, and the set of *lower priority neighbors*, denoted by $\Gamma_{-}(A_i)$ and $\Gamma_{+}(A_i)$ respectively. Formally,

$$\Gamma_{-}(A_i) \triangleq \{A_j \in \Gamma(A_i) : A_j \prec A_i\} \text{ and } \Gamma_{+}(A_i) \triangleq \{A_j \in \Gamma(A_i) : A_i \prec A_j\}.$$

Since communication between agents is not assumed to be necessarily FIFO, each agent maintains a counter that is incremented whenever the agent's value is changed. Thus, the current value of the counter *tags* each value assignment.

Definition 5.4 *An* assignment *for an agent* $A_i \in \mathcal{A}$ *is a* 3*-tuple* (q_i, v_i, t_i)*, where* $v_i \in D_i$ *and* t_i *is the tag value. When two assignments of the same variable are compared, the* most up to date *is the one that has the highest tag. Two sets of assignments,*

$$\{(q_{i_1}, v_{i_1}, t_{i_1}), \ldots, (q_{i_n}, v_{i_n}, t_{i_n})\}$$

and

$$\{(q_{j_1}, v_{j_1}, t_{j_1}), \ldots, (q_{j_m}, v_{j_m}, t_{j_m})\},$$

are compatible *if and only if every variable that appears in both sets is assigned the same value, i.e.,*

$$(\forall k,\, k \in \mathbb{N}_{\leq n})\ [(\exists! l,\, l \in \mathbb{N}_{\leq m},\, i_k = j_l) \Rightarrow (v_{i_k} = v_{j_l})].$$

Definition 5.5 *A set of asignments* $\{(q_{i_1}, v_{i_1}, t_{i_1}), \ldots, (q_{i_k}, v_{i_k}, t_{i_k})\}$ *is* consistent *if the combination of values of the variables involved satisfy every constraint involving those variables, i.e.,*

$$(\forall i_n,\, i_n \in \mathbb{N}_{\leq k})(\forall i_m,\, i_m \in \mathbb{N}_{\leq k} \setminus \{i_n\})\ (C_{i_n i_m} \in \mathcal{C} \Rightarrow (v_{i_n}, v_{i_m}) \in C_{i_n i_m})$$

Now we can define the notion of *nogood* which will prove useful in the development of the algorithm used for solving DisCSPs.

Definition 5.6 *A* nogood *is a logical conjunction of individual assignments which has been found* inconsistent.

Example 5.1. The following expression is a nogood: $\neg[(q_{i_1} = v_{i_1}) \wedge \ldots \wedge (q_{i_n} = v_{i_n})]$ which implies that the set of assignments that compose it is not consistent, meaning that the variables involved cannot attain these values without violating a constraint.

Definition 5.7 *A* directed nogood *excluding a value* $q_k \in D_k$ *as an assignment for* q_k *is an implication of the form* $[(q_{i_1} = v_{i_1}) \wedge \ldots \wedge (q_{i_n} = v_{i_n})] \rightarrow (q_k \neq v_k)$*, suggesting that the assignment* $q_k = v_k$ *is inconsistent with the assignments* $(q_{i_1} = v_{i_1}), \ldots, (q_{i_n} = v_{i_n})$*. For a nogood (ng) given in this form, we denote the* left-hand side *and* right-hand side *of the implication with respect to the position of* $\rightarrow$*, by* `lhs`*(ng) and* `rhs`*(ng) respectively.*

In order to solve a DisCSP, agents exchange their assignments with other agents.

Definition 5.8 *The* AgentView *of an agent* $A_i \in \mathcal{A}$ *is an array containing the most up to date assignments received from other agents.*

Remark 5.1. Usually, an agent is not aware of the assignments of every other agent.In such cases, the corresponding elements of AgentView are set to empty.

5.2.2 The Asynchronous Backtracking (ABT) family

There exist multiple algorithms that can be utilized for solving DisCSPs. For our purposes, we restrict our attention to the *Asynchronous Backtracking* (ABT) family [29] introduced in [30] and extended in [31]. Further algorithms as well as an in-depth analysis of DisCSPs can be found in [32]. This family of algorithms owns a set of desirable properties, namely,

(i) Agents act *concurrently*, yet *asynchronously*, based on local knowledge;

(ii) Information diffusion is limited resulting in increased *privacy*;

(iii) ABT algorithms are *sound* and *complete*.

In the ABT context, concurrency means that each agent can decide on the value of its variable without waiting for the decisions of others. However, agents are subject to a *total order*, that induces a notion of *priority*, as already described.

In particular, each agent tries to find an assignment satisfying its respective constraints with whatever knowledge it has acquired from higher-priority neighbors. Accordingly, as soon as an agent assigns a value to its variable, it communicates this assignment to lower priority neighbors. When no consistent assignment is possible for a given agent, the inconsistency is communicated other higher-priority neighbors as a (directed) nogood (Algorithm 1).

The aforementioned total ordering, and the subsequently induced partitioning of each agent's set of neighbors, corresponds to a directed flow of assignments from agents of higher priority to those of lower priority, in a relative sense. ABT family algorithms converge to a solution or detect that no solution exist in *finite time*.

In ABT algorithms, each agent retains some amount of information related to the problem at hand, in the form of two data structures, namely an $AgentView$ (5.8) and a $NoGoodStore$. The $AgentView$ consists of the most up to date assignments of higher-priority agents available, while $NoGoodStore$ is a collection of nogoods that describe inconsistencies caused by assignments of higher-priority neighbors and assignments of the agent's variable.

Handling of nogoods ABT algorithms handle nogoods using the following two principles:

- Each agent maintains at most one nogood for every value of its domain. If there exist more than one possible nogoods for a particular element of the domain set, the best nogood is selected according to the *highest possible lowest variable* heuristic [33]. As a result, an agent A_i stores at most $|D_i|$ nogoods;

- When every element of the domain set is ruled out by some nogoods, a new nogood is generated in the following manner. Let A_i be the agent under consideration and let q_j be the closest variable(in the total order sense) present in the left-hand side of some of the nogoods, with an assigned value of v_j. Then the new nogood has on its left-hand side the conjunction of all the assignments present in the the left-hand sides of nogoods where the assignment $q_j = v_j$ is present. The right-hand side is simply $q_j \neq v_j$.

  Example 5.2. A simple example of the aforementioned procedure is illustrated in Figure 5.1b - 5.1c.

The ABT family consists of multiple different algorithms. In the context of this work however, we will discuss only one, namely, adopting the nomenclature of [29], the ABT algorithm.

The ABT Algorithm Agents running the ABT algorithm use the following data structures to exchange data:

ok?: An agent sends its assignment to a *lower priority* agent;

ngd: An agent informs a *higher-priority* agent of a new nogood;

adl: An agent requests a *lower-priority* agent to set up a communication link;

stp: The problem is infeasible.

The algorithm executed by each agent $A_i \in \mathcal{A}$ is presented in Algorithm 1 in the form of pseudo-code.

Initially, agent A_i assigns a value $v_i \in D_i$ to its variable, and communicates this assignment to every lower priority neighbor. Then, a loop is executed that receives and processes messages according to their type. Once a new assignment is received, as an **ok?** message, the `ProcessInfo()` (line 22) procedure is executed, which updates the agent's $AgentView$ and subsequently performs a consistency check to determine whether the currently assigned value v_i violates any constraints (`checkAgentView()`, line 24). If v_i is inconsistent with the assignments of higher priority agents, A_i attempts to find a consistent assignment for its variable. During this process, some elements of D_i may be deemed inconsistent. Such inconsistencies are recorded in the form of nogoods which are stored in the agent's $NoGoodStore$. If a consistent value exists, it is assigned to q_i. Then, A_i sends **ok?** messages containing the new assignment to every agent in $\Gamma_+(A_i)$. Otherwise, agent A_i backtracks (line 31).

It is worth elaborating a bit further on the `UpdateAgentView()` procedure (line 25), which is called as soon as new information concerning the status of a variable becomes available. Specifically, in order to compensate for the fact that messages are not received in the order they were sent, the perceived value for a variable is seldom updated with the newly acquired value, if the the *tag* of the received assignment is *greater* than the tag of the currently adopted assignment (line 26). If the value is indeed updated, all nogoods that contained the previous assignment on their left-hand side are rendered *incompatible* with the current $AgentView$; hence they are removed from the $NoGoodStore$ (line 30).

Whenever every value in D_i is forbidden by some nogood in $NoGoodStore$, procedure `Backtrack()` is called (line 21). In order to backtrack, an agent initially resolves its $NoGoodStore$, creating a new nogood (line 32), following the procedure described in "Handling of nogoods". If the new nogood is empty the problem is infeasible, and an **stp** message is sent to every agent (line 33). In any other case, the newly generated nogood is sent as a **ng** message to the agent appearing in its *right-hand side* (line 35). Then, the assignment corresponding to the recipient of the **ng** message is removed from the $AgentView$ (line 36) and the agent selects a new, consistent value for its variable (line 37).

Whenever A_i receives a **ng** message, the `ResolveConflict()` (line 38) procedure is called. The nogood attached to the **ngd** message is accepted only if its left-hand side is compatible with the assignments (see 5.4) on the $AgentView$ of agent A_i (line 39). Next, the `CheckAddLink()` procedure (line 45) is called. This procedure adds the assignments of agents, that are not directly linked with agent A_i, and are present in the left-hand side of the received nogood, to the $AgentView$ of A_i (line 47), and requests the establishment of a communication link with those agents by sending then an **adl** message (line 49). Naturally, the

received nogood is stored, justifying thus the exclusion of the value on its right-hand side (line 41). A new consistent value for A_i is then searched (line 42), if the current value was removed by the received nogood. If the received nogood is not compatible with the $AgentView$ it is discarded. However, if the value of q_i in the right-hand aside assignment is *consistent* with the actual value of q_i, A_i sends an **ok?** message to the sender of the **ng** message containing the same assignment (line 44).

When an **adl** message is received, procedure `AddLink()` is called (line 50). Then, the sender is added in $\Gamma_+(A_i)$ (line 51). Afterwards, A_i sends its assignment through an **ok?** message to the sender if its value is different than the one included in the received **adl** message (line 52).

Example We now present a step-by-step execution of the ABT algorithm for the simple DisCSP problem illustrated in Figure 5.1. The problem includes 3 agents, i.e., $\mathcal{A} = \{A_1, A_2, A_3\}$, each controlling one variable—q_1, q_2, q_3 respectively—with their respective domains being $D_1 = \{1, 2\}$, $D_2 = \{2\}$ and $D_3 = \{1, 2\}$. The constraints of the problem are $q_1 \neq q_3$ and $q_2 \neq q_3$.

Initially, higher-priority agents, A_1 and A_2, send **ok?** messages conveying their assignments to agent A_3. Thus, the $AgentView$ of A_3 is $[(q_1 = 1), (q_2 = 2)]$. These assignments are conflicting with the values 1 and 2 of D_3. As a result two nogoods are generated and stored, namely $[q_1 = 1] \rightarrow (q_3 \neq 1)$ and $[q_2 = 2] \rightarrow (q_3 \neq 2)$(ref). Since all possible assignments for q_3 are inconsistent, agent A_3 resolves its $NoGoodStore$ by generating a new nogood $[q_1 = 1 \rightarrow q_2 \neq 2]$. This nogood is then send to agent A_2 as a **ng** message. Upon receiving this **ng** message, agent A_2 records this nogood. The assignment contained in the left-hand side of this nogood, concerns agent A_1 which is not a neighbor of agent A_2. Accordingly, agent A_2 sends an **adl** message to agent A_1, requesting the establishment of a communication link (ref). This is followed by a consistency check of its assignment with its $AgentView$. Agent A_2 must now check the consistency of its assignment. Its $AgentView(q_1 = 1)$ and its $NoGoodStore$, consisting of the single nogood received by agent A_3, result in no valid assignment, so agent A_2 resolves its $NoGoodStore$, creating the nogood $[] \rightarrow q_1 = 1$. Agent A_2 sends this nogood to agent A_1. In turn, agent A_1 change its assignment to the value 2 and sends **ok?** messages to agents A_2 and A_3. In the same manner, agent A_2 assigns a value to its variable and sends an **ok?** message to agent A_3. Finally, agents A_3 can now make a valid assignment.

Algorithm 1 ABT algorithm

```
1: procedure ABT()
2:     v_i ← empty; t_i ← 0; end ← false;
3:     CheckAgentView();
4:     while ( ¬end ) do
5:         msg ← getMsg();
6:         switch ( msg.type ) do
7:             case ok?
8:                 ProcessInfo(msg);
9:             case ngd
10:                ResolveConflict(msg);
11:            case adl
12:                AddLink(msg);
13:            case stp
14:                end ← true;
15: procedure CheckAgentView(msg)
16:    if ( ¬consistent(v_i, AgentView) ) then
17:        v_i ← ChooseValue();
18:        if ( v_i ≠ empty ) then
19:            foreach ( child ∈ Γ_+(A_i) ) do
20:                sendMsg : ok?(myAssig⟨x_i, v_i, t_i⟩) to child;
21:        else Backtrack();
22: procedure ProcessInfo(msg)
23:    UpdateAgentView(AgentView, msg.assig);
24:    CheckAgentView();
25: procedure UpdateAgentView(newAssig)
26:    if ( newAssig.tag > AgentView[j].tag ) then        % x_j ∈ newAssig %
27:        AgentView[j] ← newAssig.value;
28:        foreach ( nogood ∈ NoGoodStore ) do
29:            if ( ¬Compatible(lhs(nogood), AgentView) ) then
30:                remove(nogood, NoGoodStore);
31: procedure Backtrack()
32:    newNoGood ← solve(NoGoodStore);
33:    if newNoGood = empty then end ← true; sendMsg: stp(system);
34:    else
35:        sendMsg: ngd(newNogood) to A_j;        % x_j is the variable on rhs(newNogood) %
36:        UpdateAgentView(x_j ← empty);
37:        CheckAgentView();
```

Algorithm 1 ABT algorithm (cont'd)

38: **procedure** ResolveConflict(msg)
39: **if** (Compatible(lhs($msg.nogood$), $AgentView$)) **then**
40: CheckAddLink($msg.nogood$);
41: add($msg.nogood$, $NogoodStore$);
42: CheckAgentView();
43: **else if** (rhs($msg.nogood$).$value = v_i$) **then**
44: sendMsg : **ok?**($myAssig\langle x_i, v_i, t_i\rangle$) **to** $msg.sender$;
45: **procedure** CheckAddLink($nogood$)
46: **foreach** ($x_j \in$ lhs($nogood$) $\setminus \Gamma_-(A_i)$) **do**
47: add($x_j = v_j$ $AgentView$);
48: $\Gamma_-(A_i) \leftarrow \Gamma_-(A_i) \cup \{x_j\}$;
49: sendMsg: **adl**($x_j = v_j$) **to** A_j;
50: **procedure** AddLink(msg)
51: add($msg.sender$, $\Gamma_+(A_i)$);
52: **if** ($v_i \neq msg.assig.value$) **then** sendMsg : **ok?**($myAssig\langle x_i, v_i, t_i\rangle$) **to** $msg.sender$;
53: **function** ChooseValue(msg)
54: **foreach** ($v \in \mathcal{D}(x_i)$) **do**
55: **if** (isConsistent(v, $AgentView$)) **then return** v;
56: **else** store the best *nogood* for v;
return empty;

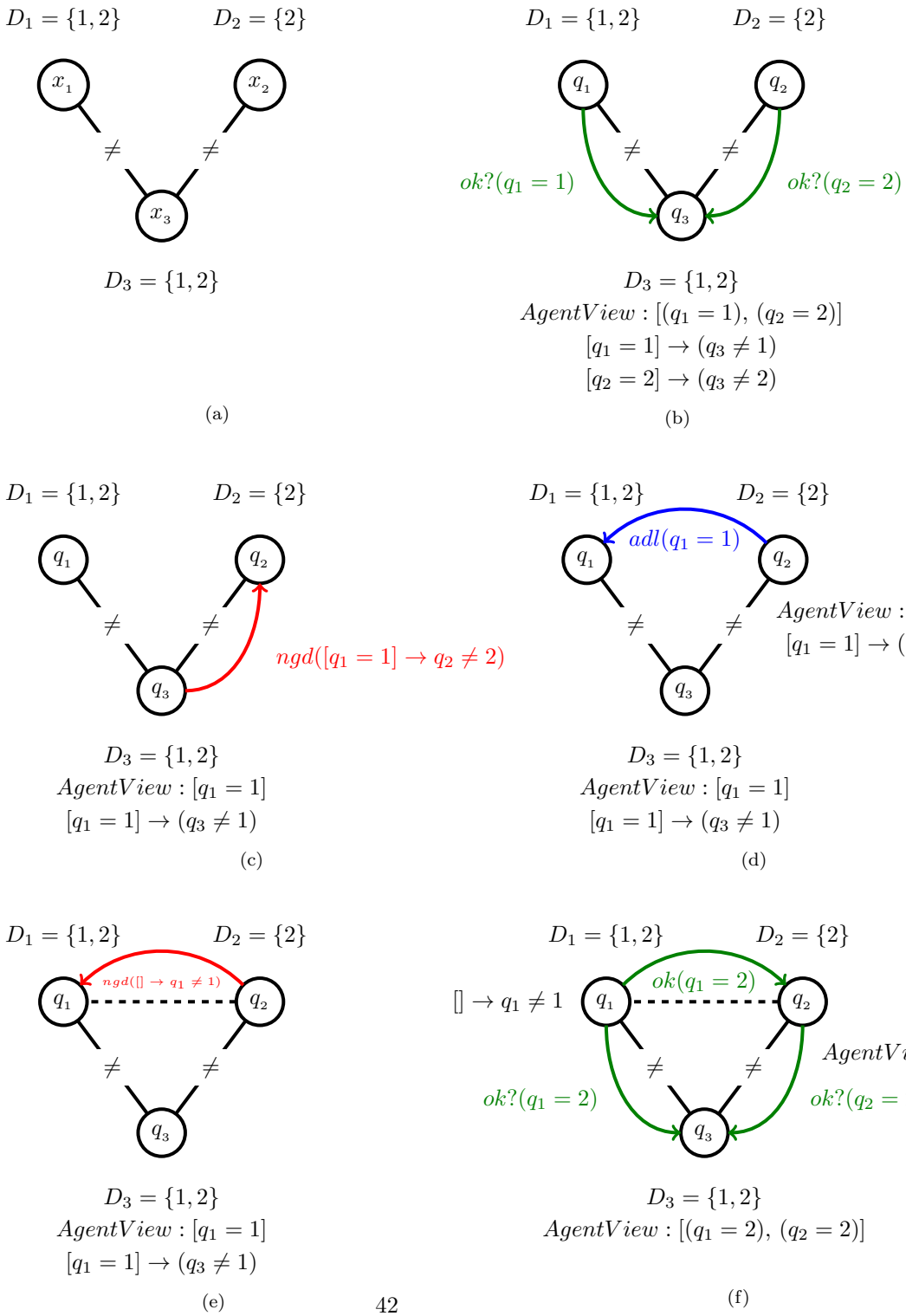


Figure 5.1: Execution example of a DisCSP.

5.3 Limited-range Delaunay Graph

We now give the necessary definitions of the notions of the *r-limited Voronoi partition* and its dual notion the *r-limited Delaunay graph.*

Definition 5.9 *Let $\mathcal{P} = \{x_i\}_{i\in\mathcal{V}}$ be a set of $n \in \mathbb{N}$ distinct points in $\mathbb{R}^2$ indexed by $\mathcal{V}$. The* Voronoi partition *[16], generated by $\mathcal{P}$ with respect to some $\|\cdot\|$ norm is the collection of sets:*

$$\{\mathbb{V}_i(\mathcal{P})\}_{i\in\mathcal{V}},$$

where,

$$\mathbb{V}_i(\mathcal{P}) \triangleq \{z \in \mathbb{R}^2 : \|z - x_i\| \leq \|z - x_j\|, \forall j \in \mathcal{V} \setminus \{i\}\}.$$

Similarly,

Definition 5.10 *Let $\mathcal{P} = \{x_i\}_{i\in\mathcal{V}}$ be a set of $n \in \mathbb{N}$ distinct points in $\mathbb{R}^2$ indexed by $\mathcal{V}$. The* r-limited Voronoi partition *[34] is the collection of sets*

$$\mathbb{V}_i^r(\mathcal{P}) \triangleq \mathbb{V}_i(\mathcal{P}) \cap \mathbb{B}\left(x_i, \frac{r}{2}\right)$$

.

Accordingly, the dual notion of r-limited Delaunay graph is defined.

Definition 5.11 *The* r-limited Delaunay graph *is a proximity graph*

$$\mathcal{G}_{LD}(\mathcal{P}, r) \triangleq (\mathcal{V}, \mathcal{E}_{LD}),$$

where the set $\mathcal{E}_{LD}$ consists of edges $(i, j) \in \mathcal{V} \times \mathcal{V}$ with the property that,

$$\mathbb{V}_i^r(\mathcal{P}) \cap \mathbb{V}_j^r(\mathcal{P}) \neq \emptyset.$$

5.4 Reconfiguration Algorithm

Once system (2.1) under the controller (4.6) reaches a stable configuration at some time τ, a check is performed to determine whether the equilibrium is the desired configuration[2]. Initially, the ρ_c-limited Delaunay graph is generated in a decentralized fashion, i.e., each agent i builds the set $\mathcal{N}_i(\mathcal{G}_{LD}(x(\tau), \rho_c))$ solely using *locally available information.* The aforementioned DisCSP,

$$\mathfrak{P} = (\mathcal{V}, \mathcal{Q}, \mathcal{D}(x(\tau)), \mathcal{C}(x(\tau), \mathcal{G}_f(\tau))),$$

is completely determined by its sets of domains and constraints. In particular, the set of domains is constructed according to Algorithm 3 which is executed in a sequential manner by each agent.

Notice that, the domain of each follower is taken as a subset of the positions of its neighbors at time τ with respect to the ρ_c-limited Delaunay graph. If the region between two agents is obstructed by obstacles and the edge corresponding to these agents in the ρ_c-limited Delaunay graph is not critical, the corresponding points are not included in the respective domains. The domain

Algorithm 3

$\mathcal{G}_t \leftarrow \mathcal{G}_{LD}(x(\tau), \rho_c)$
for each $i \in \mathcal{V}$ **do**
  for each $j \in \mathcal{N}_i(\mathcal{G}_t)$, $j > i$ **do**
    if $\mathbb{V}_i^{\rho_c}(x) \cap \mathbb{V}_j^{\rho_c}(x) \cap \mathbb{F} = \emptyset$ **then**
      if (i, j) is not a critical edge of $\mathcal{G}_t$ **then**
        $\mathcal{E}_t \leftarrow \mathcal{E}_t \setminus \{(i, j)\}$
  if $i = 1$ **then**
    $D_i \leftarrow \frac{\rho_c}{2}\left(\frac{x_1^d - x_1(\tau)}{\|x_1^d - x_1(\tau)\|}\right)$
  else
    $D_i \leftarrow \{x_j(\tau) : j \in \mathcal{N}_i(\mathcal{G}_t)\}$

of the leader is the singleton set of the point that is ρ_c away from the position of the leader towards its corresponding desired point.

The constraints C_{ij} are selected as:

- $\{(q_i, q_j) : \rho_c > \|q_i - q_j\| > 2\rho_\alpha\}$, for $j \in \mathcal{N}_i(\mathcal{G}_f(\tau))$;
- $\{(q_i, q_j) : \|q_i - q_j\| > 2\rho_\alpha\}$, for $j \in \mathcal{N}_i(\mathcal{G}_t)$;
- $\{(q_i, q_j) : \|q_i - q_j\| > 2\rho_\alpha\}$, for $j \in \bigcup_{k \in \mathcal{N}_i(G_t)} \mathcal{N}_k(\mathcal{G}_t)$;
- $D_i \times D_j$, otherwise.

Notice that the imposed constraints reflect the requirements for collision avoidance and connectivity maintenance for a pair of agents possessing a formation specification.

Therefore, Problem $\mathfrak{P}$ can be regarded as the discrete problem of assigning positions to the followers, as the leader moves closer to its desired configuration, so that agents do not overlap and agents with formation constraints are sufficiently close.

By construction and definition, it holds that

$$\mathcal{G}_t \subseteq \mathcal{G}_{LD}(x(\tau), \rho_c)$$

and

$$\mathcal{G}_{LD}(x(\tau), \rho_c) \subseteq \mathcal{G}_c(x(\tau)),$$

respectively. Thus, the maximum distance in the communication graph of agents sharing some constraint is two, a fact that substantially reduces the amount of multi-hop communication required for solving the DisCSP.

[2] Agents can agree that the system has reached an equilibrium through a consensus procedure [6].

The reconfiguration algorithm is presented in Algorithm 4. Initially, the ABT algorithm is used to solve problem $\mathfrak{P}$. During this procedure agents sharing constraints exchange their assignments and check whether those assignments violate some corresponding constraint. In case a solution is found, the relative formation specifications are updated accordingly. In case no solution exists for problem $\mathfrak{P}$, the system still acquires some useful information, i.e., how many times each constraint corresponding to a connectivity maintenance specification $(i,j) \in \mathcal{E}_f(\tau)$ was violated during the solving procedure, denoted by $\mathfrak{C}_{ij}$. It can then use this information to relax some formation specification. Through a Max-Consensus procedure [35], the specification with the highest $\mathfrak{c}_{ij}$ is chosen. Then a check is performed to determine whether the corresponding edge in the formation graph is not critical. In such case, the formation specification is removed from the graph and the system continues its motion with the reconfigured formation graph $\mathcal{G}_f(\tau')$. Alternatively, if the edge is critical, the same procedure is repeated until some non-critical edge is removed.

Algorithm 4 Reconfiguration Procedure

$(\mathcal{Q}, \{\mathfrak{C}_{\mathrm{ij}}\}_{(i,j)\in\mathcal{E}_f(\tau)}) \leftarrow \mathrm{ABT}(\mathfrak{P})$
if $\mathfrak{P}$ has a solution **then**
    $\mathcal{E}_f(\tau') \leftarrow \mathcal{E}_f(\tau)$
    for each $(i,j) \in \mathcal{E}_f(\tau')$ **do**
        $c_{ij}(\tau') \leftarrow (q_i - q_j)$
    $\mathcal{C}_f(\tau') \leftarrow \bigcup_{(i,j)\in\mathcal{E}_f(\tau')} c_{ij}(\tau')$
    $\mathcal{G}_f(\tau') \leftarrow (\mathcal{V}, \mathcal{E}_f(\tau'), \mathcal{C}_f(\tau'))$
    return $\mathcal{G}_f(\tau')$
else
    $I_f \leftarrow \mathcal{E}_f(\tau)$
    while $\{\mathfrak{C}_{ij}\}_{(i,j)\in I_f} \neq \{\emptyset\}$ **do**
        $e \leftarrow \underset{(i,j)\in I_f}{\operatorname{argmax}} \mathfrak{C}_{ij}$
        if e is not a critical edge of $\mathcal{G}_f(\tau)$ **then**
            $\mathcal{E}_f(\tau') \leftarrow \mathcal{E}_f(\tau) \setminus \{e\}$
            $\mathcal{C}_f(\tau') \leftarrow \bigcup_{(i,j)\in\mathcal{E}_f(\tau')} c_{ij}(\tau)$
            $\mathcal{G}_f(\tau') \leftarrow (\mathcal{V}, \mathcal{E}_f(\tau'), \mathcal{C}_f(\tau'))$
            return $\mathcal{G}_f(\tau')$
        else
            $I_f \leftarrow I_f \setminus \{e\}$

Note that, after a finite number of reconfigurations the leader will either reach its desired position or the formation graph will become a tree, permitting no further removal of formation specifications.

6 Line Graph Prescribed Performance Controller

6.1 Introduction

Although the reconfiguration procedure developed previously used in conjunction with the DNFs controller may lead to the desired configuration, no formal proof of convergence can be acquired. Thus, in search of a provably correct algorithm, we introduce a novel controller based on the Prescribed Performance Control (PPC) methodology [36] that can handle a subset of Problem 2.1 and specifically the case where the formation graph is a *line graph*, with the system's leader at the one end. It is reminded that following exhaustive reconfiguration, the formation graph may not be a line rather some tree. However, a line formation graph can be acquired by utilizing the DNFs controller, to increase the connectedness of the communication graph, resulting in the appearance of a line subgraph.

6.2 Prescribed Performance Control

The prescribed performance notion was originally employed to design neuroadaptive controllers, for various classes of nonlinear systems, namely feedback linearizable [36], strict feedback [37] and general MIMO affine in the control [38], capable of guaranteeing output tracking with prescribed performance. In this work, by prescribed performance, it is meant that the output tracking error converges to a predefined arbitrarily small residual set with convergence rate no less than a certain predefined value.

In that respect, consider a generic scalar tracking error $e(t)$. Prescribed performance is achieved if $e(t)$ evolves strictly within a predefined region that is bounded by certain functions of time. The mathematical expression of prescribed performance is given, $\forall t \geq 0$, by the following inequalities:

$$\varrho_l(t) < e(t) < \varrho_u(t)$$

where $\varrho_l(t), \varrho_u(t)$, are smooth and bounded functions of time satisfying

$$\lim_{t\to\infty} \varrho_l(t) < \lim_{t\to\infty} \varrho_u(t)$$

, called performance functions. The aforementioned statements are clearly illustrated in Figure 6.1 for exponential performance functions

$$\varrho_i(t) = (\varrho_{i,0} - \varrho_i^\infty)e^{-k_s t} + \varrho_i^\infty \tag{6.1}$$

with $\varrho_{i,0}, \varrho_i^\infty, k_s,\ i \in \{l, u\}$ appropriately chosen constants.

The constants $\varrho_{l,0} = \varrho_l(0)$, $\varrho_{l,0} = \varrho_l(0)$ are selected such that

$$\varrho_{l,0} < e(0) < \varrho_{u,0}.$$

The constants $\varrho_l^\infty = \lim_{t\to\infty} \varrho_l(t)$, $\varrho_u^\infty = \lim_{t\to\infty} \varrho_u(t)$ represent the maximum allowable size of the tracking error $e(t)$. Moreover, the decreasing rate of $\varrho_l(t)$,

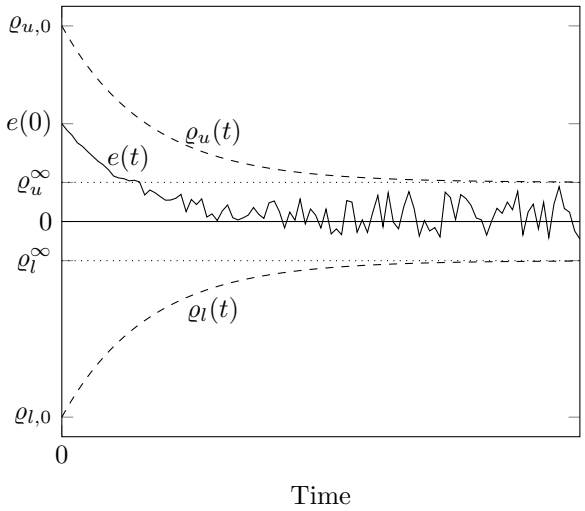


Figure 6.1: A graph illustrating the Prescribed Performance Control methodology.

$\varrho_l(t)$ which is affected by the constant k_s in this case, introduces a lower bound on the required speed of convergence of $e(t)$. Therefore, the appropriate selection of the performance functions $\varrho_l(t)$, $\varrho_u(t)$ imposes performance characteristics on the tracking error $e(t)$.

Instead of choosing performance functions that are functions of time, we will utilize dynamic performance functions that are governed by differential equations driven by state-dependent terms. This approach, under certain prevailing assumptions will allow us to derive a convergent and safe controller for formations graphs that are line graphs with the leader at one end.

6.3 Modelling

For each agent $i \in \mathcal{V}$, we define the distance from the leader as the bijection,

$$\text{dist}_1 : \mathcal{V} \to \mathbb{N}_{<n},\ i \mapsto \text{dist}(i, 1).$$

Each follower $i \in \mathcal{V}_f \triangleq \mathcal{V} \setminus \{1\}$, is assigned the desired relative formation specification

$$c_i \triangleq c_{ij},$$

where

$$j = \text{dist}_1^{-1}(\text{dist}_1(i) - 1).$$

Equivalently, the configuration of j with respect to i can be expressed in polar form as, $d_i \triangleq \|x_j - x_i\|$ and $\theta_i \triangleq \text{atan2}(x_j - x_i)$. The equivalent formulation for the desired formation configuration is then set as $d_i^d \triangleq \|c_i\|$ and $\theta_i^d \triangleq \text{atan2}(c_i)$.

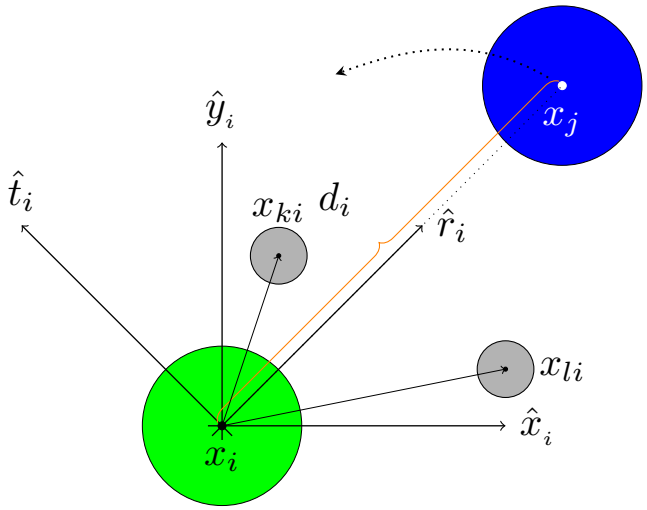


Figure 6.2: The $(\hat{x}_i, \hat{y}_i)$ frame results from translating the global frame to the center of agent i. The orthonormal frame $(\hat{r}_i, \hat{t}_i)$ is fixed at x_i with the first coordinate pointing towards agent j at all times. In this example, $k \in {}^iI_{o,l}$ and $l \in {}^iI_{o,r}$. Informally, the two classes separate the obstacles to those on the left of and those on the right of $\hat{r}_i$.

By the path homotopy property, the set of obstacles I_o can be partitioned into two classes with respect to each follower $i \in \mathcal{V}_f$, namely

$$ {}^iI_{o,l} \triangleq \{k \in I_o : x_{ki} \times x_{kj} > 0\},$$

and

$$ {}^iI_{o,r} \triangleq \{k \in I_o : x_{ki} \times x_{kj} < 0\},$$

where the "$\times$" operator is the 2-dimensional cross product and $x_{ki} \triangleq x_k - x_i$ (see Figure 6.2). We are now ready to proceed with the formulation of the Prescribed Performance based controller.

6.4 Controller Design

We will apply the Prescribed Performance Approach separately for the errors of the distance d_i and angle θ_i from the desired value. Accordingly, we define four performance functions governed by the following set of linear differential equations:

$$\dot{\varrho}_{\theta_i,u} = -k_s(\varrho_{\theta_i,u} - \varrho^{\infty}_{\theta_i} - \theta^d_i) + f_{\theta_i}, \tag{6.2a}$$

$$\dot{\varrho}_{\theta_i,l} = -k_s(\varrho_{\theta_i,l} + \varrho^{\infty}_{\theta_i} - \theta^d_i) + f_{\theta_i}, \tag{6.2b}$$

$$\dot{\varrho}_{d_i,u} = \mathrm{Proj}_{\Pi}(-k_s(\varrho_{d_i,u} - \varrho^{\infty}_{d_i} - d^d_i) + f_{d_i}; \varrho_{d_i,u}), \tag{6.2c}$$

$$\dot{\varrho}_{d_i,l} = \mathrm{Proj}_{\Pi}(-k_s(\varrho_{d_i,l} + \varrho^{\infty}_{d_i} - d^d_i) + f_{d_i}; \varrho_{d_i,l}), \tag{6.2d}$$

where $\varrho_{\theta_i}^{\infty}$, $\varrho_{d_i}^{\infty} \in \mathbb{R}_{>0}$ are the *ultimate steady state error bounds* for θ_i and d_i respectively, and $k_s \in \mathbb{R}_{>0}$ is the *convergence rate*. Moreover, $Proj_{\Pi}(\cdot\,;\cdot)$ denotes a projection operator [39] which ensures that the corresponding state variables remain in the convex set $\Pi \triangleq \mathbb{R}_{>2\rho_\alpha}$, thus guaranteeing inter-agent collision avoidance.

Remark 6.1. The homogeneous solution of (6.2a) - (6.2d), i.e., for $f = 0$, is identical to the exponential decaying performance function defined in (6.1).

The input terms are defined as:

$$f_{\theta_i} \triangleq f_{i,l} - f_{i,r}, \tag{6.3a}$$

$$f_{d_i} \triangleq \left(\frac{\sigma_{\epsilon_5, r_p}^2}{1 - \sigma_{\epsilon_5, r_p}^2} \right) \left(\max \left\{ f_{i,l}, f_{i,r} \right\} \right), \tag{6.3b}$$

where,

$$f_{i,l} \triangleq \max_{k \in {}^{i}I_{o,l}} \left\{ \frac{1 - b_{ijk}}{b_{ijk}}, \frac{1 - b_{ik}}{b_{ik}} \right\}, \tag{6.4a}$$

$$f_{i,r} \triangleq \max_{k \in {}^{i}I_{o,r}} \left\{ \frac{1 - b_{ijk}}{b_{ijk}}, \frac{1 - b_{ik}}{b_{ik}} \right\}, \tag{6.4b}$$

where b_{ijk} and b_{ik} have been assigned in [].

The role of those terms is to adequately temporarily alter the performance functions so that agent i can follow the leader without colliding with an obstacle or violating path homotopy.

Following the design procedure of Prescribed Performance Control, the *normalized errors* of θ_i and d_i are defined as:

$$\zeta_{\theta_i} \triangleq \frac{\theta_i - \frac{\varrho_{\theta_i,u} + \varrho_{\theta_i,l}}{2}}{\frac{\varrho_{\theta_i,u} - \varrho_{\theta_i,l}}{2}}, \tag{6.5}$$

and

$$\zeta_{d_i} \triangleq \frac{d_i - \frac{\varrho_{d_i,u} + \varrho_{d_i,l}}{2}}{\frac{\varrho_{d_i,u} - \varrho_{d_i,l}}{2}}, \tag{6.6}$$

respectively. the control input for each follower is then formulated as:

$$u_i(t) = R(\theta_i) \begin{bmatrix} v_{\hat{r}_i} \\ v_{\hat{t}_i} \end{bmatrix}, \quad i \in \mathcal{V}_f, \tag{6.7}$$

where,

$$v_{\hat{r}_i} \triangleq k_r ln \left(\frac{1 + \zeta_{d_i}}{1 - \zeta_{d_i}} \right) - \frac{\zeta_{d_i} + 1}{2} \dot{\rho}_{d_i,u} + \frac{\zeta_{d_i} - 1}{2} \dot{\rho}_{d,l}, \tag{6.8}$$

and,

$$v_{\hat{t}_i} \triangleq d_i \left(k_t ln \left(\frac{1 + \zeta_{\theta_i}}{1 - \zeta_{\theta_i}} \right) - \frac{\zeta_{\theta_i} + 1}{2} \dot{\rho}_{\theta,u} + \frac{\zeta_{\theta_i} - 1}{2} \dot{\rho}_{\theta,l} \right), \tag{6.9}$$

are the control inputs expressed in the $(\hat{r}_i, \hat{t}_i)$ frame, $k_r, k_t \in \mathbb{R}_{>0}$ are control gains. $R(\theta_i) \in SO(2)$ is the rotation matrix from the $(\hat{r}_i, \hat{t}_i)$ frame to the inertial frame $(\hat{x}_i, \hat{y}_i)$. Finally, the control input for the *leader* is provided by some controller (e.g., a Navigation Function).

6.5 Analysis

The proposed control scheme guarantees that both θ_i and d_i will be invariably upper and lower bounded by their respective performance functions, assuming this statement is initially true[3]. Then, by appropriate manipulation of the aforementioned performance functions (6.2a)-(6.2d), agent i can follow agent j maintaining path homotopy and avoiding the obstacles while at the same time preserving connectivity.

The principle of operation of the controller is based on the observation that in the proximity of obstacles belonging to the same class (i.e., either ${}^iI_{o,l}$ or ${}^iI_{o,r}$), all terms b_{ijk} and b_{ik} are increased by changing the angle θ_i towards the opposite direction, thus moving away from the obstacles. This is achieved through the term f_{θ_i} in (6.2a) and (6.2b) which induces the appropriate control command in the direction of $\hat{t}_i$ (6.9). In the proximity of obstacles that belong to both classes, the aforementioned action may not suffice. Based on the fact that reducing the distance d_i leads to an increase of every b_{ijk}, additional control command in the $\hat{r}_i$ direction (6.8), is induced by the f_{d_i} term in (6.2c) and (6.2d). An indicative case is presented in Figure 6.2, where while changing the angle θ_i in an attempt to avoid violation of path homotopy with respect to obstacle k, agent i risks collision with obstacle l. Then reducing the distance d_i, obstacle k is bypassed and the angle action can be utilized for avoiding collision with obstacle l. Away from obstacles, the terms f_{θ_i} and f_{d_i} are zero and thus, the homogeneous terms of (6.2a)-(6.2d) bound agent i towards the desired configuration, as the performance functions asymptotically converge towards an arbitrarily small neighborhood (dictated by $\varrho_{\theta_i}^{\infty}$ and $\varrho_{d_i}^{\infty}$) of the desired configuration.

This is summarized in the following theorem:

Theorem 6.1 *Given a constant line formation graph, and an initial configuration that respects path homotopy* (3.10)*, system* (2.1) *under the control law* (6.7) *will converge to an arbitrarily small neighborhood of the desired configuration, without collisions or loss of connectivity as long as the leader has bounded velocity.*

Proof. The analysis can be performed for a system of one leader and one follower and then applied recursively to the rest of the followers. Collisions are handled by the projection operator and since the d_i is non-increasing connectivity cannot be jeopardized. Terms (6.3a), (6.3b) tend to infinity as some b_{ijk} or b_{ik} tends to zero. By the path homotopy assumption, simultaneous violation

[3]This a valid assumption given that both θ_i and d_i are initially known, rendering thus proper initialization trivial.

of path homotopy specifications that belong to different classes cannot occur since that would lead to a contradiction, i.e., a path homotopy violation would have previously occurred. Thus, performance functions can be altered in such a way, that the agent can follow its leader. Since the leader has bounded velocity, the agent can always develop a sufficiently high, yet bounded velocity to remain within the bounds dictated by the performance functions. Away from obstacles, the values of the performance functions return close to the desired. □

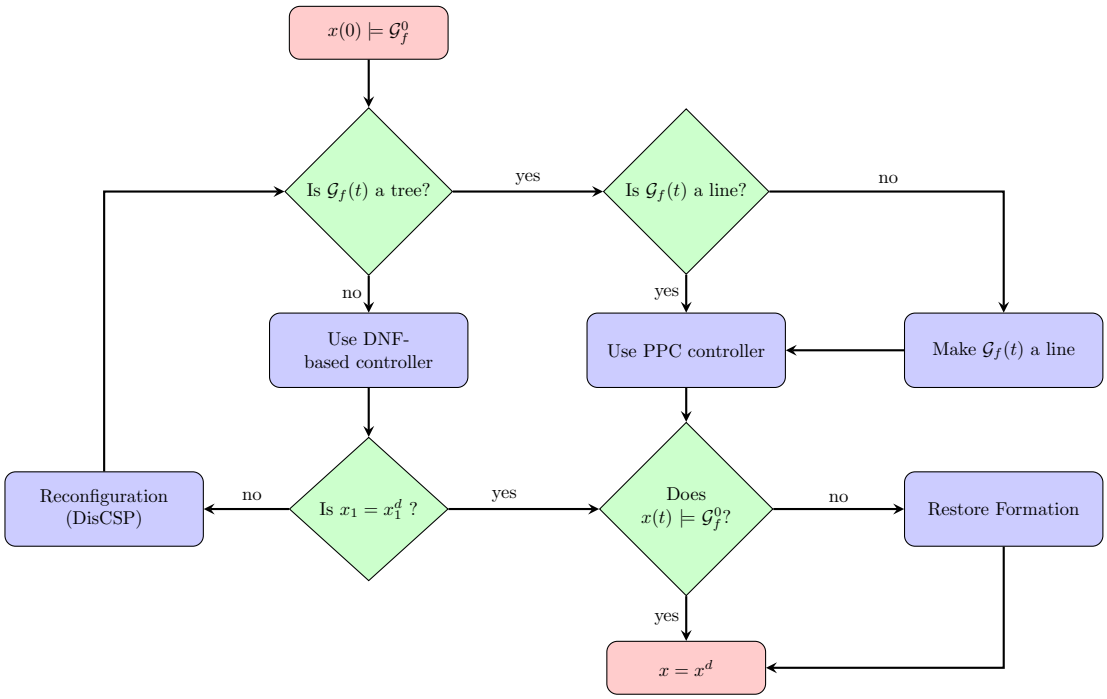


Figure 7.1: Flow chart of the integrated scheme.

7 Integrated Scheme

The flow diagram describing the integrated scheme is presented in Figure 7.1 Finally, note that the system can successfully attain the initial formation graph, once the leader has reached its desired point, by reversing the reconfiguration process. This is achieved using the DNFs controller with guaranteed convergence, since, by Assumption 4, obstacles are sufficiently away and the analysis reduces to [23].

8 Simulation Results

We now present some simulation results that validate our approach. The continuous controllers were programmed using the C programming language, while the reconfiguration algorithm and the integration was implemented in MATLAB running on an Intel Core i7 2630QM with 4 GB RAM. The execution time for the whole simulation was in the order of a few seconds.

Three cases scenarios are presented. The first requires no reconfiguration, the second converges after one reconfiguration and the third converges using the PPC based controller.

The numerical values that follow were common in all 3 cases.

System properties

- $\rho_\alpha = 1$ (agents radius)
- $\rho_c = 10$ (communication and sensing radius)

DNFs based controller parameters

- $\kappa = 1000$ (DNF parameter)
- $\epsilon_1 = 0.1$ (Workspace containment buffer parameter)
- $\epsilon_2 = 0.1$ (Collision avoidance buffer parameter)
- $\epsilon_3 = 0.1$ (Connectivity maintenance buffer parameter)
- $R = \sqrt{\rho_c^2 - (2\rho_\alpha)^2}$ (Maximum allowed distance between formation graph neighbors)
- $\epsilon_4 = 0.1$ (Path homotopy maintenance buffer parameter)

Line graph PPC contoller

- $\varrho_{\theta_i}^{\infty} = 0.1$ (θ_i ultimate steady state error bound)
- $\varrho_{d_i}^{\infty} = 0.1$ (d_i ultimate steady state error bound)
- $k_s = 1$ (Error convergence rate)
- $\epsilon_5 = 0.1$ (Switch transition zone width)
- $r_p = 10$ (Switch threshold)
- $k_r = 4$ (Radial direction control gain)
- $k_t = 4$ (Tangential direction control gain)

8.1 Case 1

The first case involves four agents and is resolved without reconfiguration through the DNFs based controller. Details are now provided.

Setup

- Number of agents: 4
- Number of obstacles: 6
- $\rho_w = 60$
- $x_1(0) = [-17,\ -36]^T$
- $x_1^d = [17,\ 20]^T$

Formation Specifications

- $c_{12} = [3.5355,\ 3.5355]^T$
- $c_{13} = [-3.5355,\ 3.5355]^T$
- $c_{14} = [0,\ 7.0711]^T$
- $c_{23} = [-7.0711,\ 0]^T$
- $c_{24} = [-3.5355,\ 3.5355]^T$
- $c_{25} = [0,\ 7.0711]^T$
- $c_{34} = [3.5355,\ 3.5355]^T$
- $c_{36} = [0,\ 7.0711]^T$
- $c_{45} = [3.5355,\ 3.5355]^T$
- $c_{46} = [-3.5355,\ 3.5355]^T$
- $c_{56} = [-7.0711,\ 0]^T$
- $c_{57} = [0,\ 7.0711]^T$
- $c_{68} = [0,\ 7.0711]^T$
- $c_{78} = [-7.0711,\ 0]^T$

Obstacles

k	x_k	ρ_k
1	$[-24.5,\ -30]^T$	3
2	$[-16,\ -30]^T$	3
3	$[-14,\ -15]^T$	6
4	$[5,\ -17]^T$	9
5	$[-14,\ 0]^T$	6
6	$[5,\ 4]^T$	9

8.2 Case 2

In this scenario the system consists of eight agents. The system gets stuck at a local minimum configuration. However, following the removal of a formation specification, the system successfully converges to the desired configuration.

Setup

- Number of agents: 8
- Number of obstacles: 12
- $\rho_w = 60$
- $x_1(0) = [-17, -36]^T$
- $x_1^d = [15.7, 32]^T$

Formation Specifications

- $c_{12} = [3.5355, 3.5355]^T$
- $c_{13} = [-3.5355, 3.5355]^T$
- $c_{14} = [0, 7.0711]^T$
- $c_{23} = [-7.0711, 0]^T$
- $c_{24} = [-3.5355, 3.5355]^T$
- $c_{25} = [0, 7.0711]^T$
- $c_{34} = [3.5355, 3.5355]^T$
- $c_{36} = [0, 7.0711]^T$
- $c_{45} = [3.5355, 3.5355]^T$
- $c_{46} = [-3.5355, 3.5355]^T$
- $c_{56} = [-7.0711, 0]^T$
- $c_{57} = [0, 7.0711]^T$
- $c_{68} = [0, 7.0711]^T$
- $c_{78} = [-7.0711, 0]^T$

Obstacles

k	x_k	ρ_k	k	x_k	ρ_k
1	$[-16, -30]^T$	3	7	$[-33, -15]^T$	6
2	$[-14, -15]^T$	6	8	$[-43, -15]^T$	6
3	$[5, -17]^T$	9	9	$[-51, -15]^T$	6
4	$[14, -3]^T$	7	10	$[20, -17]^T$	9
5	$[5, 1]^T$	9.5	11	$[35, -17]^T$	9
6	$[-23, -15]^T$	6	12	$[46, -17]^T$	9

8.3 Case 3

In the third scenario the system consists again of eight agents. The workspace is more cluttered than in the previous case. Thus, after a series of reconfigurations the formation graphs ends up being a line graph. The system then switches to the PPC controller and successfully converges to the desired configuration.

Setup

- Number of agents: 8
- Number of obstacles: 30
- $\rho_w = 62$
- $x_1(0) = [-17,\ -36]^T$
- $x_1^d = [26.2,\ 44.4]^T$

Formation Specifications

- $c_{12} = [3.5355,\ 3.5355]^T$
- $c_{13} = [-3.5355,\ 3.5355]^T$
- $c_{14} = [0,\ 7.0711]^T$
- $c_{23} = [-7.0711,\ 0]^T$
- $c_{24} = [-3.5355,\ 3.5355]^T$
- $c_{25} = [0,\ 7.0711]^T$
- $c_{34} = [3.5355,\ 3.5355]^T$
- $c_{36} = [0,\ 7.0711]^T$
- $c_{45} = [3.5355,\ 3.5355]^T$
- $c_{46} = [-3.5355,\ 3.5355]^T$
- $c_{56} = [-7.0711,\ 0]^T$
- $c_{57} = [0,\ 7.0711]^T$
- $c_{68} = [0,\ 7.0711]^T$
- $c_{78} = [-7.0711,\ 0]^T$

Obstacles

k	x_k	ρ_k	k	x_k	ρ_k	k	x_k	ρ_k
1	$[-16, -30]^T$	3	11	$[48, -17]^T$	9	21	$[-7.5, 14.4]^T$	0.4
2	$[-14, -17]^T$	6	12	$[5, 1]^T$	9.5	22	$[-10.3, 14.4]^T$	0.4
3	$[-24, -17]^T$	6	13	$[-7.5, -8]^T$	2	23	$[-13.1, 14.4]^T$	0.4
4	$[-34, -17]^T$	6	14	$[-7.5, -5.2]^T$	0.4	24	$[-15.9, 14.4]^T$	0.4
5	$[-44, -17]^T$	6	15	$[-7.5, -2.4]^T$	0.4	25	$[-18.7, 14.4]^T$	0.4
6	$[-52, -17]^T$	6	16	$[-7.5, 0.4]^T$	0.4	26	$[-21.5, 14.4]^T$	0.4
7	$[5, -17]^T$	9	17	$[-7.5, -3.2]^T$	0.4	27	$[-24.3, 14.4]^T$	0.4
8	$[15, -17]^T$	9	18	$[-7.5, 5.6]^T$	0.4	28	$[-27.1, 14.4]^T$	0.4
9	$[25, -17]^T$	9	19	$[-7.5, 8.8]^T$	0.4	29	$[-29.9, 14.4]^T$	0.4
10	$[35, -17]^T$	9	20	$[-7.5, 11.6]^T$	0.4	30	$[-32.7, 14.4]^T$	0.4

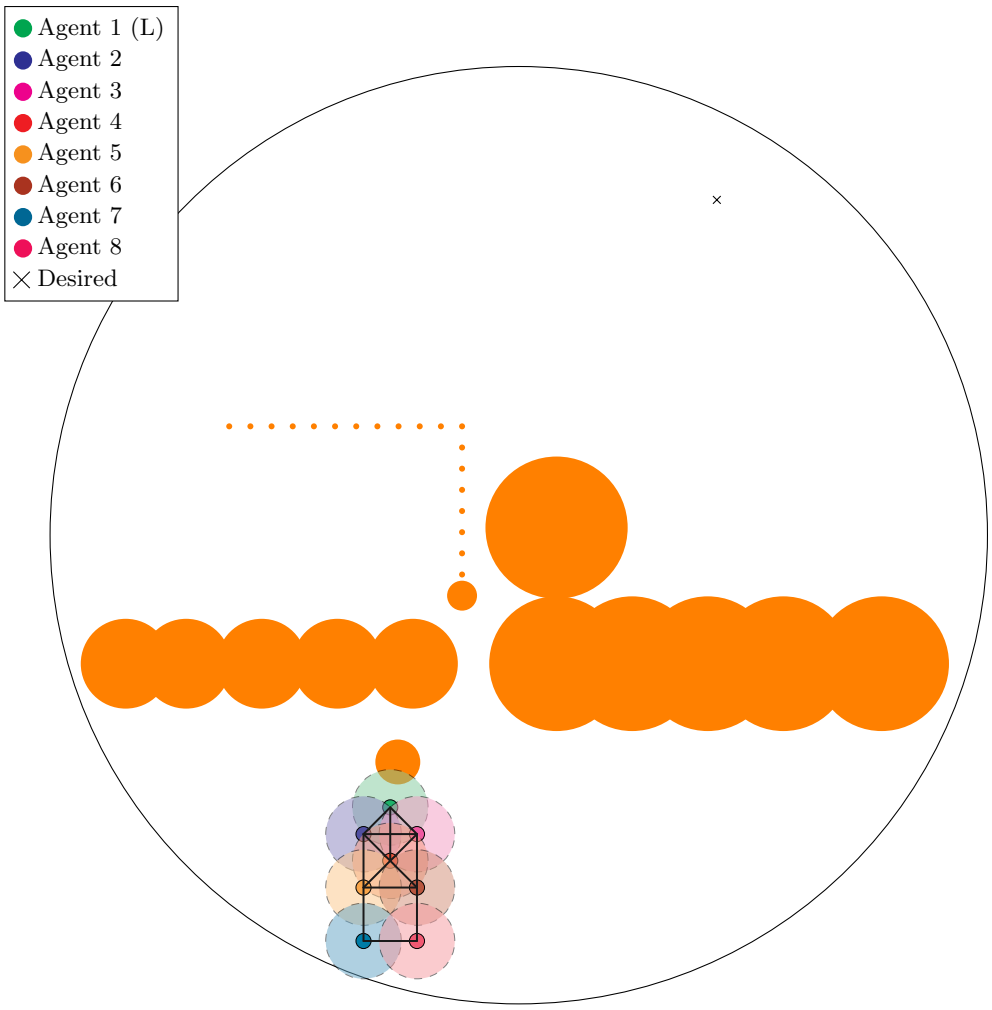


Figure 8.1: At the initial configuration the system starts operating with the DNFs controller.

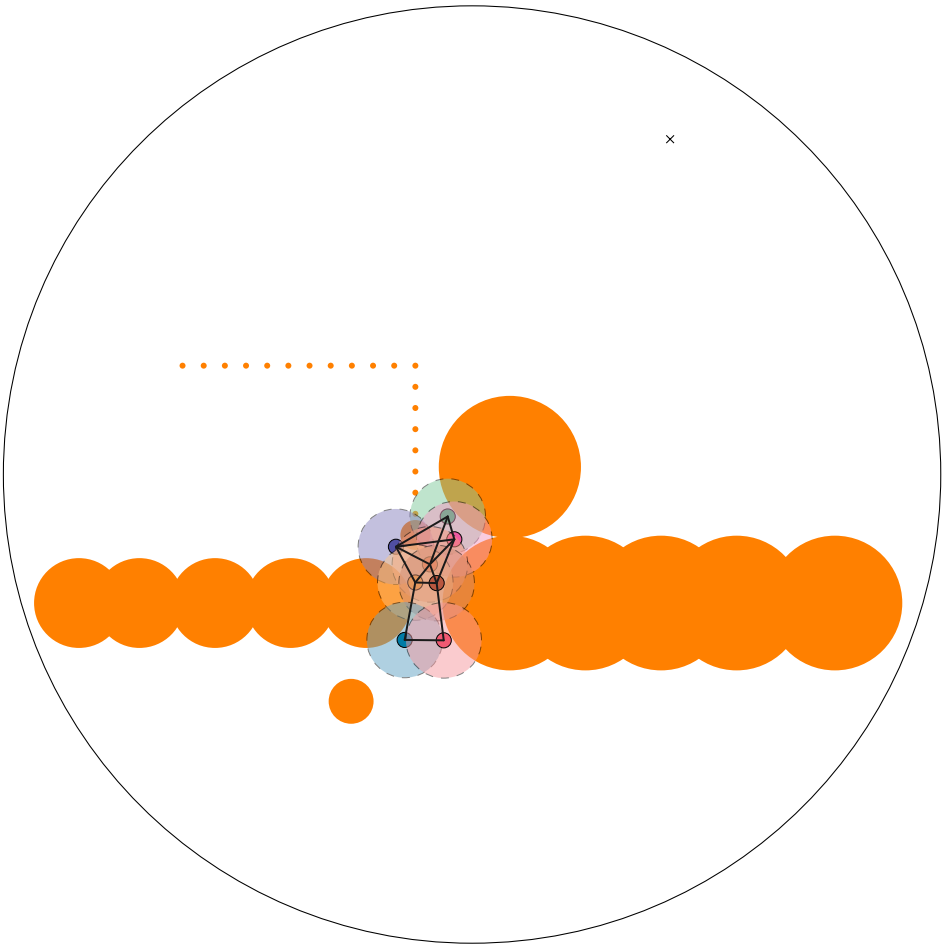


Figure 8.2: System encounters the first unwanted local minimum.

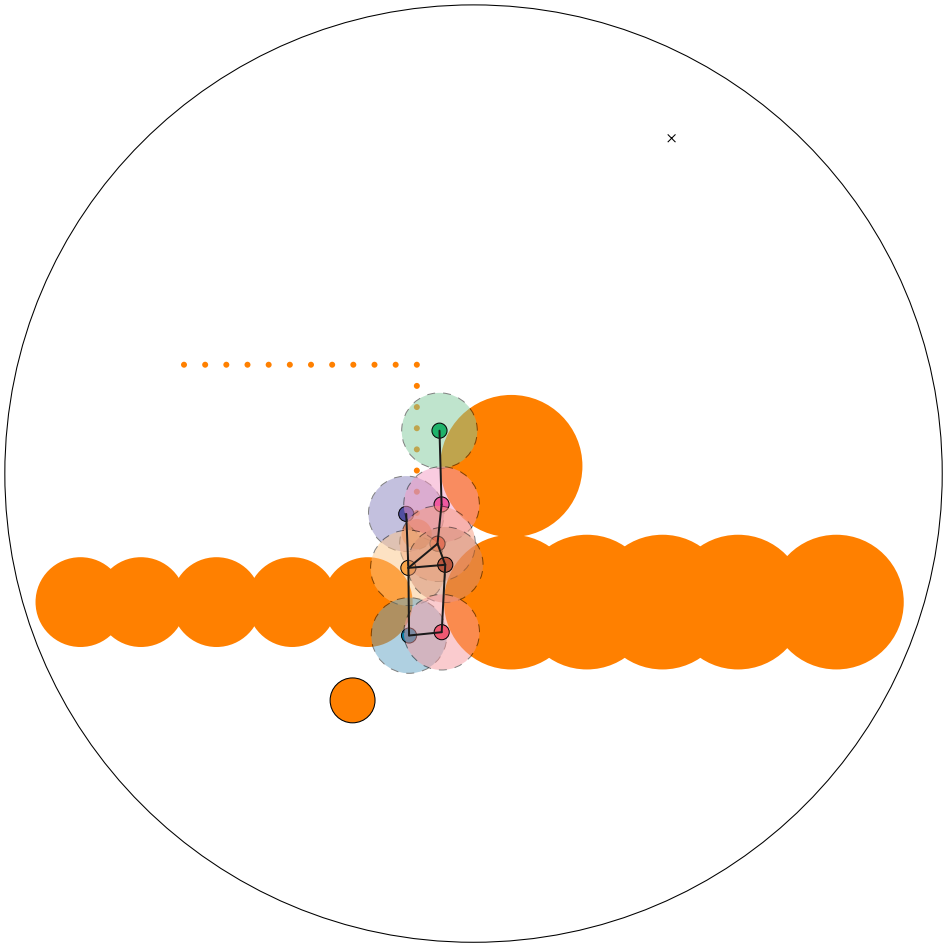


Figure 8.3: The system makes substantial progress towards the desired configuration.

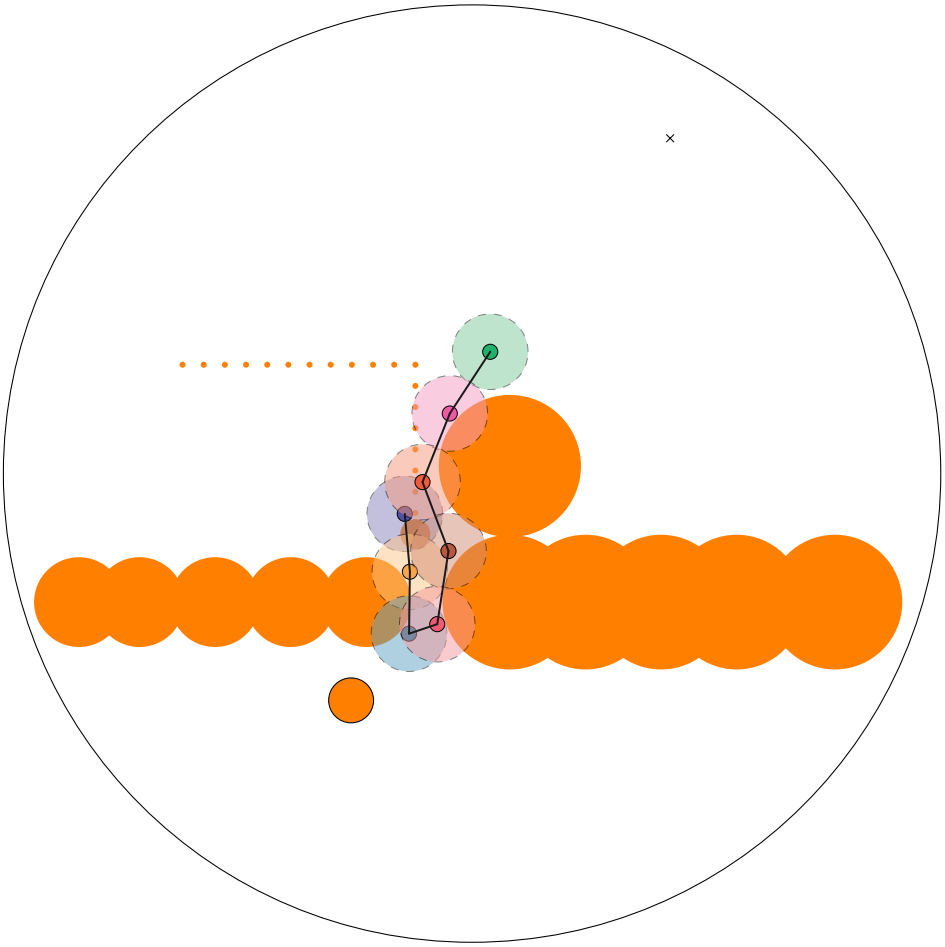


Figure 8.4: Additional reconfiguration result in a line formation graph and controller switch is performed.

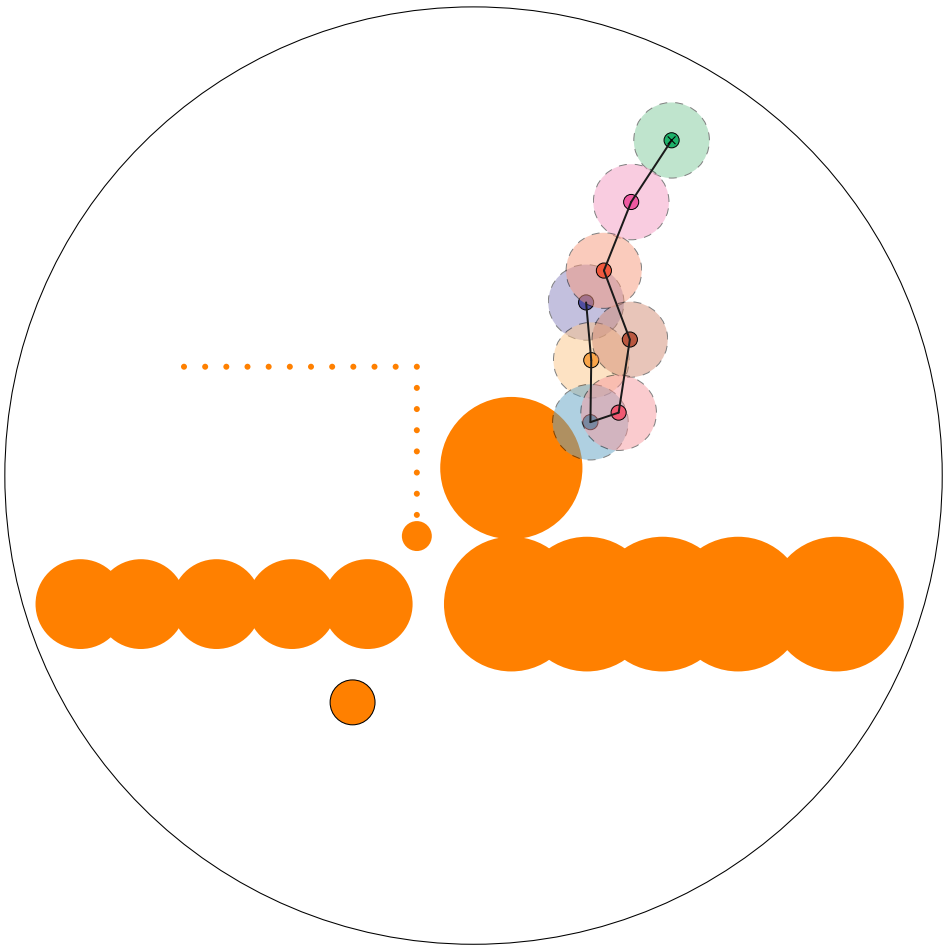


Figure 8.5: All obstacles are successfully negotiated, and the leader reaches its desired point.

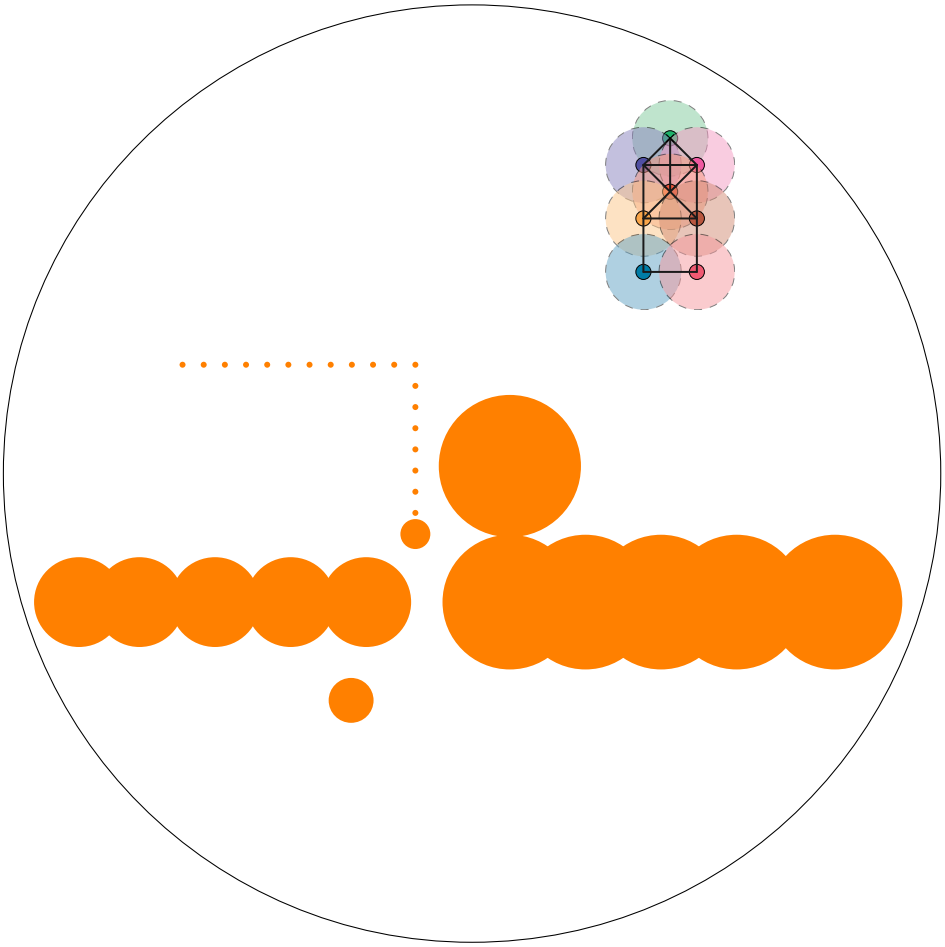


Figure 8.6: Using the DNFs based controller desired formation is attained as well.

9 Future Research

In the following, we propose some directions for further research that are related to the problem we addressed and the notions we utilized.

Adaptive DNFs based controller The DNFs based controller developed in subsection 4.1 suffered from persistent local minima. It would be interesting to see whether some inner control loop of the relative formation specifications, i.e., c_{ij}'s, could alleviate those issues. In this task, the path homotopy notion could prove useful, similarly to the case of the controller presented in section 6.

Extension of the PPC controller to tree graphs The PPC controller proposed in can at the moment handle seldom line formation graphs. However, it is our belief that the aforementioned controller could be extended to handle general tree graphs. This could perhaps be achieved with the aid of a supervisory discrete controller that would swap edges, i.e., add one formation specification and subsequently remove another. Note that in this case no criticality check is necessary.

Predict In our approach the system had to reach a halt, i.e., a local minima before the reconfiguration procedure was initiated. Perhaps some adaptive control scheme, such as the ones proposed above, combined with a less sophisticated discrete decision algorithm would deliver better results. In those lines, a distributed edge criticality detection would prove extremely useful.

References

[1] I. D. Couzin, J. Krause, N. R. Franks, and S. A. Levin. Effective leadership and decision-making in animal groups on the move. *Nature*, 433(7025):513–516, 2005.

[2] A. Jadbabaie, J. Lin, and A. S. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, 48(6):988–1001, 2003.

[3] J. A. Fax and R. M. Murray. Information flow and cooperative control of vehicle formations. *IEEE Transactions on Automatic Control*, 49(9):1465–1476, 2004.

[4] R. Olfati-Saber and R. M. Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on Automatic Control*, 49(9):1520–1533, 2004.

[5] W. Ren and R. W. Beard. Consensus seeking in multiagent systems under dynamically changing interaction topologies. *IEEE Transactions on Automatic Control*, 50(5):655–661, 2005.

[6] R. Olfati-Saber, J. A. Fax, and R. M. Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, Jan 2007.

[7] H. G. Tanner, A. Jadbabaie, and G. J. Pappas. Flocking in fixed and switching networks. *Automatic Control, IEEE Transactions on*, 52(5):863–868, may 2007.

[8] Maria Carmela De Gennaro and Ali Jadbabaie. Formation control for a cooperative multi-agent system using decentralized navigation functions. In *American Control Conference, 2006*, pages 6–pp. IEEE, 2006.

[9] R.O. Saber and R.M. Murray. Flocking with obstacle avoidance: cooperation with limited communication in mobile networks. In *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, volume 2, pages 2022 – 2028 Vol.2, dec. 2003.

[10] P Ogren. Split and join of vehicle formations doing obstacle avoidance. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 2, pages 1951–1955. IEEE, 2004.

[11] M. Colledanchise, D. V. Dimarogonas, and P. Ogren. Obstacle avoidance in formation using navigation-like functions and constraint based programming. In *IEEE International Conference on Intelligent Robots and Systems*, pages 5234–5239, 2013.

[12] C. P. Bechlioulis and K. J. Kyriakopoulos. Robust model-free formation control with prescribed performance for nonlinear multi-agent systems. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, submitted.

[13] C. P. Bechlioulis and K. J. Kyriakopoulos. Robust model-free formation control with prescribed performance and connectivity maintenance for nonlinear multi-agent systems. In *Proceedings of the IEEE Conference on Decision and Control*, 2014, submitted.

[14] D. V. Dimarogonas and K. H. Johansson. Decentralized connectivity maintenance in mobile networks with bounded inputs. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1507–1512, May 2008.

[15] J. M. Esposito and T. W. Dunbar. Maintaining wireless connectivity constraints for swarms in the presence of obstacles. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 946–951, May 2006.

[16] Francesco Bullo, Jorge Cortes, and Sonia Martinez. *Distributed Control of Robotic Networks: A Mathematical Approach to Motion Coordination Algorithms*. Princeton University Press, Princeton, NJ, USA, 2009.

[17] Z. Kan, A. P. Dani, J. M. Shea, and W. E. Dixon. Network connectivity preserving formation stabilization and obstacle avoidance via a decentralized controller. *IEEE Transactions on Automatic Control*, 57(7):1827–1832, 2012.

[18] Daniel E. Koditschek and Elon Rimon. Robot navigation functions on manifolds with boundary. *Adv. Appl. Math.*, 11(4):412–442, December 1990.

[19] James R. Munkres. *Topology: A First Course*. Prentice Hall, 1975.

[20] Sergey V. Matveev. *Lectures on Algebraic Topology (EMS Series of Lectures in Mathematics) (English and Russian Edition)*. European Mathematical Society, April 2006.

[21] Dimos Dimarogonas, Savvas Loizou, and Kostas Kyriakopoulos. Multi-robot navigation functions ii: towards decentralization. *Stochastic Hybrid Systems*, pages 209–253, 2006.

[22] Morris William Hirsch, Stephen Smale, and Robert Luke Devaney. *Differential equations, dynamical systems, and an introduction to chaos*. Academic Press, Waltham (Mass.), 2013.

[23] D.V. Dimarogonas and E. Frazzoli. Analysis of decentralized potential field based multi-agent navigation via primal-dual lyapunov theory. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 1215–1220, Dec 2010.

[24] Daniel E Koditschek and Elonlate Rimon. Robot navigation functions on manifolds with boundary. *Adv. Appl. Math.*, 11(4):412–442, December 1990.

[25] Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38(3):353 – 366, 1989.

[26] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):673–685, Sep 1998.

[27] M. Yokoo. *Distributed Constraint Satisfaction: Foundations of Cooperation in Multi-Agent Systems*. Springer Series on Agent Technology. Springer-Verlag Berlin Heidelberg, 1st edition, 2001.

[28] Bernd Schroder. *Ordered Sets: An Introduction with Connections from Combinatorics to Topology*. Birkhauser, 2nd edition, 2016.

[29] C. Bessiere, Ismel Brito, Arnold Maestre, and Pedro Meseguer. The asynchronous backtracking family. Technical Report TR 03139, LIRMM-CNRS, March 2003.

[30] M. Yokoo, T. Ishida, E. H. Durfee, and K. Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *Distributed Computing Systems, 1992., Proceedings of the 12th International Conference on*, pages 614–621, Jun 1992.

[31] Christian Bessiere, Arnold Maestre, Ismel Brito, and Pedro Meseguer. Asynchronous backtracking without adding links: a new member in the abt family. *Artificial Intelligence*, 161(1):7 – 24, 2005.

[32] Mohamed Wahbi. *Algorithms and Ordering Heuristics for Distributed Constraint Satisfaction Problems*. Focus Series. John Wiley & Sons, Inc., 2013.

[33] Katsutoshi Hirayama and Makoto Yokoo. The effect of nogood learning in distributed constraint satisfaction. *2013 IEEE 33rd International Conference on Distributed Computing Systems*, 0:169, 2000.

[34] K. R. Guruprasad and P. Dasgupta. Distributed voronoi partitioning for multi-robot systems with limited range sensors. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3546–3552, Oct 2012.

[35] F. Iutzeler, P. Ciblat, and J. Jakubowicz. Analysis of max-consensus algorithms in wireless channels. *IEEE Transactions on Signal Processing*, 60(11):6103–6107, Nov 2012.

[36] C. P. Bechlioulis and G. A. Rovithakis. Robust adaptive control of feedback linearizable mimo nonlinear systems with prescribed performance. *IEEE Transactions on Automatic Control*, 53(9):2090–2099, 2008.

[37] C. P. Bechlioulis and G. A. Rovithakis. Adaptive control with guaranteed transient and steady state tracking error bounds for strict feedback systems. *Automatica*, 45(2):532–538, 2009.

[38] C. P. Bechlioulis and G. A. Rovithakis. Prescribed performance adaptive control for multi-input multi-output affine in the control nonlinear systems. *IEEE Transactions on Automatic Control*, 55(5):1220–1226, 2010.

[39] Z. Cai, M. S. de Queiroz, and D. M. Dawson. A sufficiently smooth projection operator. *IEEE Transactions on Automatic Control*, 51(1):135–139, Jan 2006.

A Graph Theory Notions

A graph $\mathcal{G}_* = (\mathcal{V}, \mathcal{E}_*)$ consists of a finite set of vertices $\mathcal{V}$ and a finite set of edges $\mathcal{E}_* \subset \mathcal{V} \times \mathcal{V}$. We restrict our attention to simple undirected graphs, i.e., $(i,i) \notin \mathcal{E}_*$ and $(i,j) \in \mathcal{E}_* \Leftrightarrow (j,i) \in \mathcal{E}_*$. A graph $\mathcal{G}_s = (\mathcal{V}, \mathcal{E}_s)$ is a *subgraph* of $\mathcal{G}_*$, denoted by $\mathcal{G}_s \subseteq \mathcal{G}*$, if $\mathcal{E}_s \subseteq \mathcal{E}_*$. For each vertex $i \in \mathcal{V}$ we define the set of *neighbors* $\mathcal{N}_i(\mathcal{G}_*) \triangleq \{j \in \mathcal{V} : (i,j) \in \mathcal{E}_*\}$. The degree $\deg_i(\mathcal{G}_*)$of a vertex i is the cardinality of the corresponding set of neighbors, i.e., $\deg_i(\mathcal{G}_*) \triangleq |\mathcal{N}_i(\mathcal{G}_*)|$. A (simple) path on $\mathcal{G}_*$ is a finite sequence of *unique* elements of $\mathcal{V}$ such that every two consecutive elements are *adjacent*. The length of the path is defined as the number of vertices minus one. We say that a graph is *connected* if and only if for each pair of distinct vertices there exists a path that contains both vertices. The *distance* of two vertices u, v of a connected graph, denoted by $\text{dist}(u,v)$ is equal to the length of the shortest path in the graph that contains both vertices. An edge $e \in \mathcal{E}_*$ of a connected graph $\mathcal{G}_*$ is called a *critical edge* if the graph $(\mathcal{V}, \mathcal{E}_* \setminus \{e\})$ is not connected. A *tree* is a graph with all of its edges being critical. A *line graph* is a tree with $\max_{i \in \mathcal{V}}\{\deg_i(\mathcal{G}_*)\} = 2$.

The task of determining whether an edge $e = (i,j)$ is critical is of global nature, meaning that it cannot in *general* be determined locally. In the context of this work, this is achieved using a *Distributed Breadth-First Search* algorithm, which searches for another path that contains vertices i and j other than (i,j).