UNIVERSITY OF CALIFORNIA

SANTA CRUZ

CONTROLLING GLOBAL NETWORK CONNECTIVITY OF ROBOT SWARMS WITH LOCAL INTERACTIONS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

APPLIED MATH AND STATISTICS

by

Michael D. Schuresko

March 2009

The Dissertation of Michael D. Schuresko
is approved:

Professor Jorge Cortés, Chair

Professor Herbert Lee

Professor Hongyun Wang

Professor Katia Obraczka

Dean Lisa Sloan
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Controlling Global Network Connectivity of Robot Swarms with Local Interactions

by

Michael D. Schuresko

It is common for distributed mobile robot control algorithms to operate under the constraint that individual robots communicate over a wireless network, where communication links are determined by spatial proximity among mobile agents. Many of these control algorithms make no guarantee that the robots will move in such a way as to maintain network connectivity. We perform research into developing robust efficient distributed algorithms to solve this and related problems and show progress on current work towards this end.

To Jamie,

And to the little one,

Who does not yet have a name.

Acknowledgments

I would like to thank, first my advisor, Dr Jorge Cortés, who provided just the right balance of whip-cracking and support to help me finish in a reasonable time with a publication record I am happy with. I would also like to thank my parents, Daniel and Janet Schuresko, and my aunt and uncle, Susan Dawson and Milo Puz, who were there for me in all the ways I needed it.

I would like to thank Jamie Vuong for sticking with me while I completed this ordeal, and for making life more livable towards the end of graduate school. I also thank her family for their love and understanding as we juggled a relationship and graduate school.

My friends have all been a big help in pursuing my PhD, especially Peter Towbin, whose place I am crashing at as I write this, and Karl Obermeyer and Katie Laventall who, during our rare encounters, inspire me by reminding me that "cooperative control of mobile robotics" is actually a really interesting research area. Tom Bagby, Lydia Choy and Katie Laventall all encouraged me to try Processing.org. While I never ended up using it for my research, a little playing with it inspired me with a solution for the graphics on my simulations. I would also like to thank, in no particular order, Tim Chevalier, Gabriel Elkaim, Marian Farah, Kevin Greenan, Hiroyuki Kajima, Tom Maher, Gyuri Ordody, Kevin Buoren Shiue, Quentin Todd, Walter Vannini, Yishan Wong.

Finally I would like to thank the myriad of engineers behind the Ubuntu linux distribution, the matplotlib library for Python, and the Eclipse IDE for Java, all of which have been a joy to use and made graduate school significantly easier.

Part I

Background and introduction

Chapter 1

Introduction

Recently the controls community has witnessed an explosion of distributed coordination algorithms that operate on swarms of robots.

Colloquially speaking, a swarm of robots is some number of mobile robots that each move independently, while using some form of communication to coordinate their motions. Usually the amount of communication allowed is somehow limited, both in terms of which robots can directly communicate, and how much data can be sent per unit time. A thorough survey of the general area of distributed control of robot swarms is [7]. Images of physical implementations of such swarms are shown in Figure 1.1 (courtesy [60]) and Figure 1.2 (courtesy [9]).

Much of the inspiration for research in mobile robot swarm research comes from biological systems [48, 6]. Schools of fish and flocks of birds appear to exhibit coherent behavior despite being comprised of many independent agents. Work by Craig Reynolds [48] first popularized and convincingly demonstrated the idea that artificial agents could exhibit qualitatively similar behavior to flocking birds or schooling fish. Ants colonies [6] and bee swarms [47] also exhibit interesting group behavior which can inspire algorithmic research in analogous swarms of mobile robots. Work on designing robot swarm

Figure 1.1: Robot swarm developed at MIT. Image courtesy [60], see also [61]

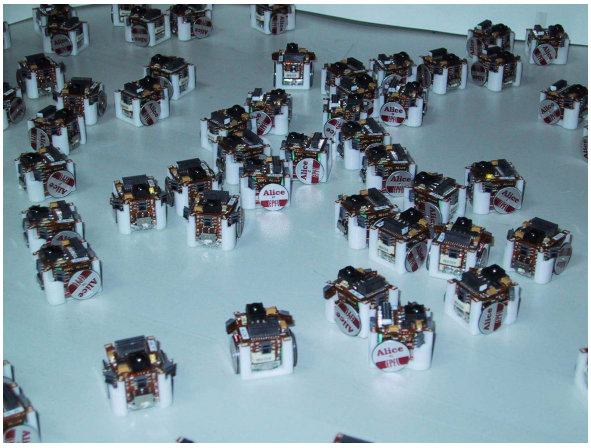


Figure 1.2: Robot swarm developed at EPFL (Lausanne, Switzerland). Image courtesy [9], see also [10, 8]

control algorithms has, in turn, fed back into the biological literature, particularly [29], which studies the collective motion of bacterial colonies.

Many of the inspirations for swarming research come from systems in which many agents interacting with simple rules give rise to a global emergent behavior. In some sense the task of designing control algorithms for swarms is the inverse problem to the common scientific task of predicting collective behavior from atomic rules of individual agents [3] (for instance how the actions of individual neurons give rise to the behavior of the brain, how individual consumers interact to create macroeconomic behavior, how organisms give rise to ecosystems). Instead of seeking models that predict which global behaviors will arise from local rules, we seek to engineer local rules to give rise to desired global behaviors.

One might hope, therefore, that work in this area may help shed light on the related scientific problem of understanding complex interactions in multi-agent systems, and help better design the sorts of engineered systems like cities and networks that mirror the sprawling many-agent nature of the systems (populations, ecosystems, markets) with which they interact.

On a more immediate level, potential applications for such work include environmental monitoring [32, 46, 49], urban search and rescue [52], but could also be extended to related areas of ad-hoc networks [38, 31, 50] and claytronics [1] (the use of swarms of tiny robots to form granular programmable materials). The proposed NASA Terrestrial Planet Finder [28], shown in Figure 1.3, would use coordination among automous robotic spacecraft to create a virtual astronomical instrument.

Work on this topic often draws on existing techniques from control theory and distributed algorithms [33] to solve a wide range of motion coordination problems for swarms of robots. Of particular interest to us are systems in which the set of robots with which a given robot can communicate are based on proximity.

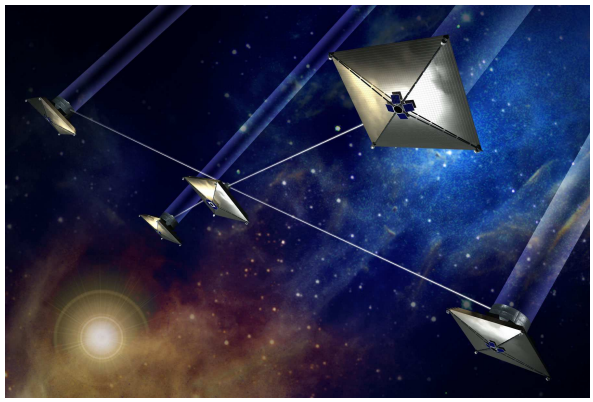


Figure 1.3: Artist's conception of the NASA Terrestrial Planet Finder (TPF). Image courtesy [40], see also [27]

Examples of algorithms to control distributed swarms of mobile agents include flocking [48, 65, 23], shown in Figure 1.4 and deployment [14], shown in Figure 1.5.

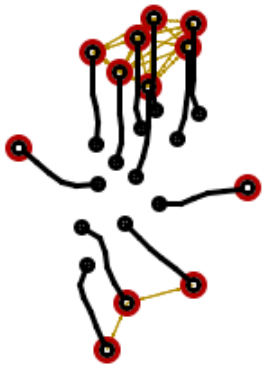


Figure 1.4: Simulation of flocking algorithm presented in [65]. Light lines are placed between robots which can communicate.

Many algorithms for the coordinated control of a swarm of robots make no guarantee that the swarm will remain connected, and not "shatter" into two or more disjoint swarms. If the network over which the robots communicate is based on proximity, such an event could cause the communication

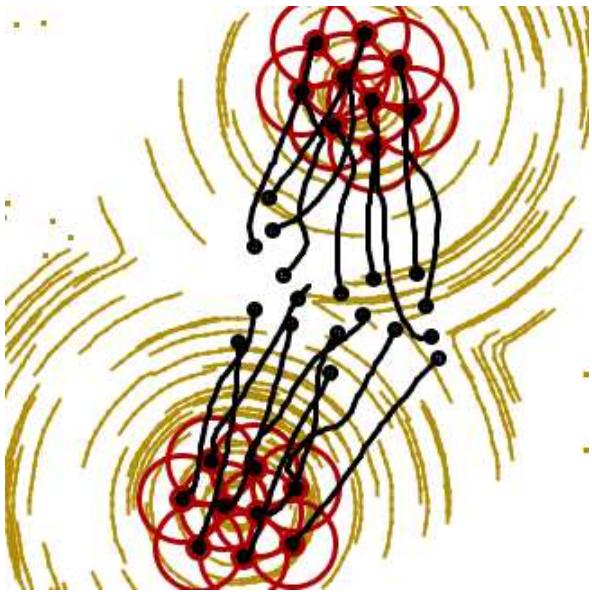


Figure 1.5: Simulation of deployment algorithm presented in [14]. Robots are maximizing the integral over the union of their sensor coverage disks (large circles) of the density field indicated by the contour lines. Also shown are agent start positions (small circles) and paths between initial and final positions (centers of large circles).

network to become disconnected. Obviously a split of this nature could cause performance degradation for many distributed motion coordination algorithms.

There are several notions of what it means for a swarm of robots to be "connected." Most of these notions have to do with the connectivity of some sort of graph induced by wireless inter-robot communication, or inter-robot sensing. It turns out that certain common robotic motion coordination tasks, such as flocking [23], [65], consensus [63], [43] and rendezvous [13], [36], [2], have performances which are characterized by measures of this communication graph connectivity.

As of yet, the existing literature for algorithms to control the connectivity of robot swarms through constraining robot motion is fairly sparse. In this work, we engage in a course of research into the possibility of richer, more flexible, efficient motion coordination algorithms to control the connectivity of robot swarms.

This thesis includes work from four published conference papers, one [54] discussing the mate-

rial in Section 4, two [55, 58] discussing the material in Section 5 and the third [56] discussing material in Section 6. This work also reflects two journal papers, the first [57] provides further details on Section 6, while the second [59] elaborates on recent developments to Section 5 first discussed in [58].

Chapter 2

Background and preliminaries

In Section 2.1 we will discuss the robotic network model we use as a framework for reasoning about distributed motion coordination algorithms. In Section 2.2 we discuss an extension of this model which is useful for formulating connectivity problems.

In Section 2.3 we review notions from algebraic graph theory which will be at the heart of one of our approaches to connectivity problems later in Section 6.

Finally Section 2.7 goes into detail on a series of connectivity tasks and what it means to solve them.

2.1 Robotic network model

We will present our algorithms within the framework introduced in [37] for synchronous robotic networks. For completeness, we present a brief account of the model here.

We start with our definition of a "Robotic Network."

[Informal description] First (in Definition 2.1.1) we describe our abstraction of the underlying physics of a swarm of robots i.e. how the robots are capable of moving and under which conditions they are capable of communicating.

Then (in Definition 2.1.2) we specify how we denote motion coordination algorithms. Each robot runs a discrete time algorithm, which can send messages to the robots neighbors in the communication graph at specified time instants. Each robot also has a continuous-time control system to control its trajectory which can read the state of the discrete time algorithm, but not directly communicate with neighboring robots.

Finally, after these definitions, we will introduce the natural notion of the "evolution" of a robotic network.

Definition 2.1.1 (Robotic network)**.** *A* robotic network $\mathcal{S}$ *is a tuple* $(I, \mathcal{A}, E_{\text{cmm}})$ *consisting of*

(i) $I = \{0, \dots, n-1\}$*; I is called the* set of unique identifiers (UIDs)*;*

(ii) $\mathcal{A} = \{A^{[i]}\}_{i \in I}$*, with* $A^{[i]} = (X^{[i]}, U^{[i]}, X_0^{[i]}, f)$*, is a set of control systems; where $X^{[i]}$ is the state-space of the ith control system and $U^{[i]}$ is the control space of the ith control system; this set is called the* set of physical agents*;*

(iii) E_{cmm} *is a map from* $\prod_{i \in I} X^{[i]}$ *to the subsets of* $I \times I \setminus \text{diag}(I \times I)$*; this map is called the* communication edge map*.*

If $A^{[i]} = (X, U, X_0, f)$ *for all* $i \in I$*, then the robotic network is called* uniform*.*

Definition 2.1.2. *A* (synchronous, static, uniform, feedback) control and communication law $\mathcal{CC}$ *for* $\mathcal{S}$ *consists of the sets:*

(i) $\mathbb{T} = \{t_\ell\}_{\ell \in \mathbb{N}_0} \subset \mathbb{R}_{\geq 0}$ *is an increasing sequence of time instants, called* communication schedule*;*

(ii) L *is a set containing the* `null` *element, called the* communication language*; elements of L are called* messages*;*

(iii) $W^{[i]} = W$*, $i \in I$, are sets of values of some* logic variables $w^{[i]}$*, $i \in I$;*

(iv) $W_0^{[i]} \subseteq W$*, $i \in I$, are subsets of* allowable initial values*;*

and of the maps:

(i) $\mathrm{msg} : \mathbb{T} \times X \times W \times I \to L$ *is called the* message-generation function*;*

(ii) $\mathrm{stf} : \mathbb{T} \times W \times L^n \to W$*, is called the* state-transition function*;*

(iii) $\mathrm{ctl} : \bar{\mathbb{R}}_+ \times X \times X \times W \times L^n \to U$*, $i \in I$, is called the* control function.

When we refer to an "evolution" of a robotic network, we mean the behavior of the network starting from a valid initial state. The execution of a control and communications law can be roughly described as follows: at each communication round, each agents sends messages to its neighbors according to the evaluation of msg. With the messages received, each agent updates the value of its logic variables using stf. In between communication rounds, the motion of each agent motion is governed by ctl. A precise description of an execution can be found in [37].

2.1.1 Concrete Example

As a contrete example of a robotic network and a control and communication law, we will present the "Agree and Pursue Law" from [37] and its associated network. It should be noted that our presentation of this example is taken almost verbatim from [37] and does not constitute an original contribution of this work.

Let $\mathbb{S}^1$ be the unit circle, and measure positions on $\mathbb{S}^1$ counterclockwise from the positive horizontal axis. For $x, y \in \mathbb{S}^1$, we let $\mathrm{dist}(x, y) = \min\{\mathrm{dist}_{\mathsf{c}}(x, y), \mathrm{dist}_{\mathsf{cc}}(x, y)\}$. Here, $\mathrm{dist}_{\mathsf{c}}(x, y) = (x - y) \,(\mathrm{mod}\, 2\pi)$ is the clockwise distance, that is, the path length from x to y traveling clockwise. Similarly, $\mathrm{dist}_{\mathsf{cc}}(x, y) = (y - x) \,(\mathrm{mod}\, 2\pi)$ is the counterclockwise distance. Here $x \,(\mathrm{mod}\, 2\pi)$ is the remainder of the division of x by 2π.

Example 2.1.3. (Locally-connected first-order agents on the circle) *For $r \in \mathbb{R}_+$, consider the uniform robotic network $\mathcal{S}_{\mathrm{circle}} = (I, \mathcal{A}, E_{r\text{-disk}})$ composed of identical agents of the form $(\mathbb{S}^1, (0, \mathbf{e}_{sph}))$.*

Here $\mathbf{e}_{sph}$ *is the vector field on* $\mathbb{S}^1$ *describing unit-speed counterclockwise rotation. We define the* r*-disk proximity edge map* $E_{r\text{-disk}}$ *on the circle by setting* $(i,j) \in E_{r\text{-disk}}(\theta^{[1]}, \ldots, \theta^{[N]})$ *if and only if*

$$\mathrm{dist}(\theta^{[i]}, \theta^{[j]}) \leq r\,,$$

where $\mathrm{dist}(x,y)$ *is the geodesic distance between the two points* x, y *on the circle.* •

We now define the agree-and-pursue law, denoted by $\mathcal{CC}_{\text{agr-pursuit}}$, as the uniform and time-independent law loosely described as follows:

> *[Informal description]* The dynamic variables are `drctn` taking values in $\{\mathtt{c}, \mathtt{cc}\}$ and `prior` taking values in I. At each communication round, each agent transmits its position and its dynamic variables and sets its dynamic variables to those of the incoming message with the largest value of `prior`. Between communication rounds, each agent moves in the counterclockwise or clockwise direction depending on whether its dynamic variable `drctn` is `cc` or `c`. For $k_{\text{prop}} \in]0, \frac{1}{2}[$, each agent moves k_{prop} times the distance to the immediately next neighbor in the chosen direction, or, if no neighbors are detected, k_{prop} times the communication range r.

Next, we define the law *formally*. Each agent has logic variables $w = (\mathtt{drctn}, \mathtt{prior})$, where $w_1 = \mathtt{drctn} \in \{\mathtt{cc}, \mathtt{c}\}$, with arbitrary initial value, and $w_2 = \mathtt{prior} \in I$, with initial value equal to the agent's identifier i. In other words, we define $W = \{\mathtt{cc}, \mathtt{c}\} \times I$, and we set $W_0^{[i]} = \{\mathtt{cc}, \mathtt{c}\} \times \{i\}$. Each agent $i \in I$ operates with the standard message-generation function, i.e., we set $L = \mathbb{S}^1 \times W$ and $\mathrm{msg}^{[i]} = \mathrm{msg}_{\text{std}}$, where $\mathrm{msg}_{\text{std}}(\theta, w, j) = (\theta, w)$. The state-transition function is defined by

$$\mathrm{stf}(w, y) = \mathrm{argmax}\{z_2 \ : \ z \in (\pi_L(y))_2 \cup \{w\}\}.$$

For $k_{\text{prop}} \in \mathbb{R}_+$, the control function is

$$\mathrm{ctl}(\theta, \theta_{\mathrm{strt}}, w, y) = k_{\mathrm{prop}} \begin{cases} \min\{r\} \cup \{\mathrm{dist}_{\mathtt{cc}}(\theta_{\mathrm{strt}}, \theta_{\mathrm{rcvd}}) \ : \ \theta_{\mathrm{rcvd}} \in (\pi_L(y))_1\}, & \text{if } \mathtt{drctn} = \mathtt{cc}, \\ -\min\{r\} \cup \{\mathrm{dist}_{\mathtt{c}}(\theta_{\mathrm{strt}}, \theta_{\mathrm{rcvd}}) \ : \ \theta_{\mathrm{rcvd}} \in (\pi_L(y))_1\}, & \text{if } \mathtt{drctn} = \mathtt{c}. \end{cases}$$

Finally, we sketch the control and communication in equivalent pseudocode language. This is possible for this example, and necessary for more complicated ones. For example, the state-transition function is written as:

```
function stf ((drctn, prior), y)
for each non-null message
        (θ_rcvd, (drctn_rcvd, prior_rcvd)) in y:
  if (prior_rcvd > prior), then
    drctn := drctn_rcvd
    prior := prior_rcvd
  endif
endfor
return (drctn,prior)
```

Similarly, the control function ctl is written as:

```
function ctl (θ, θ_strt, (drctn, prior), y)
d_tmp := r
for each non-null message
        (θ_rcvd, (drctn_rcvd, prior_rcvd)) in y:
  if (drctn = cc) AND (dist_cc(θ_strt, θ_rcvd) < d_tmp),
    then d_tmp := dist_cc(θ_strt, θ_rcvd)
  elseif (drctn = c) AND (dist_c(θ_strt, θ_rcvd) < d_tmp),
    then d_tmp := dist_c(θ_strt, θ_rcvd)
  endif
endfor
if (drctn = cc), then return k_prop d_tmp,
  else return −k_prop d_tmp endif
```

An implementation of this control and communication law is shown in Fig. 2.1. Note that, along the evolution, all agents agree upon a common direction of motion and, after suitable time, they reach a uniform distribution. Finally, we remark that this law is related to leader election algorithms, e.g., see [33], and to cyclic pursuit algorithms, e.g., see [35, 62]. •

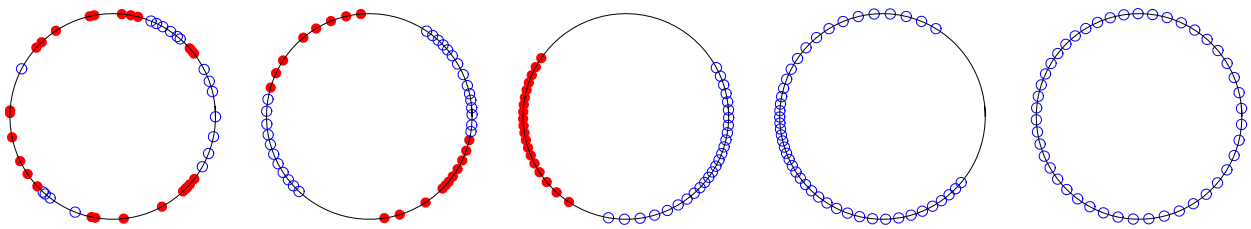


Figure 2.1: The agree-and-pursue control and communication law in Section 2.1.1 with $N = 45$, $r = 2\pi/40$, and $k_{\text{prop}} = 1/4$. Disks and circles correspond to agents moving counterclockwise and clockwise, respectively. The initial positions and the initial directions of motion are randomly generated. The five pictures depict the network state at times $0, 12, 37, 100, 400$. From [37]

2.2 Input-Output control and communication laws

To facilitate the development of reusable components which can be composed to form valid control and communication laws, we introduce here the concept of an input-output control and communication law.

> *[Purpose for this construction]* An "*input-out control and communication law*" is like a "control and communication law" (see Definition 2.1.2) with extensions which allow it to interact with another "control and communication law" running on the same robotic network. This construction allows control and communication laws to be constructed which place some essential sub-task behind a level of abstraction, allowing the mechanism that solves the sub-task to be any control and communication law following some core set of properties.

Definition 2.2.1. *A* (synchronous, static, uniform, feedback) input-output control and communication law $\mathcal{CC}$ *for a uniform network* $\mathcal{S}$ *consists of the sets:*

(i) $\mathbb{T} = \{t_\ell\}_{\ell \in \mathbb{N}_0} \subset \mathbb{R}_{\geq 0}$*, a communication schedule;*

(ii) L*, a communication language;*

(iii) (W, W_{in}, W_{out})*, sets of values of* logic variables*,* input logic variables*, and* output logic variables*,* $i \in I$*, respectively;*

(iv) $W_0^{[i]} \subseteq W$*,* $i \in I$*, subsets of* allowable initial values*;*

(v) ${W_{in,0}}^{[i]} \subseteq W_{in}$*, subsets of* allowable initial input values*;*

and of the maps:

(i) $\mathrm{msg} : X \times W \times W_{in} \times \mathbb{Z}_n \to L$*, the* message-generation function*;*

(ii) $\mathrm{stf_{io}} : W \times W_{in} \times L^n \to W \times W_{out}$ *the* (input-output) state-transition function*;*

(iii) $\mathrm{ctl} : X \times X \times W \times W_{in} \to U$*, the* control function.

Without loss of generality, unless otherwise stated, we consider $\mathbb{T} = \mathbb{Z}_{\geq 0}$. For notational convenience, we often write an input-output state-transition function stf_{io} as the pair $(\text{stf}_{\text{io}-\text{slf}}, \text{stf}_{\text{io-out}})$, where $\text{stf}_{\text{io}-\text{slf}}$ computes values in W and $\text{stf}_{\text{io-out}}$ in W_{out}. We refer to $\text{stf}_{\text{io-out}}$ as the *output state transition function*.

Note that a *control and communication law* is an input-output control and communication law with $W_{\text{in}} = \emptyset = W_{\text{out}}$.

Remark 2.2.2. We note that the algorithms presented in this paper work equally well if the slight modification is made that the control function ctl, which defines the instantaneous motion of the robot in continuous time, is replaced with a *waypoint generation function*, defining the goal position that a given robot should be at during the next communication round. Such a function can be defined like waypt : $X \times W \times W_{\text{in}} \times L^n \to X$. In particular, this observation implies that we can consider arbitrary agent dynamics so long as the requirements imposed by the waypoint generation function can be satisfied by the dynamics. •

While it may look strange to combine control and communication laws by summing the control components of each law, it should be noted that each input-output control and communication law has the freedom to set its control function to zero between any two communication rounds based on its logic variables.

A composition of two input-output laws is the natural result of substituting a subset of each law's output for a subset of the other law's input. We detail this next.

Definition 2.2.3 (Composition of input-output laws)**.** *The composition of two input-output control*

and communication laws, $\mathcal{CC}_1$ *and* $\mathcal{CC}_2$*, that satisfy*

$$\mathcal{CC}_1 W_{in} = X \times Y, \quad \mathcal{CC}_1 W_{in0} = X_0 \times Y_0, \quad \mathcal{CC}_1 W_{out} = B \times C,$$

$$\mathcal{CC}_2 W_{in} = A \times B, \quad \mathcal{CC}_2 W_{in0} = A_0 \times B_0, \quad \mathcal{CC}_2 W_{out} = Y \times Z,$$

for some sets X, Y, Z, A, B, C*, with* $A_0 \subset A$*,* $B_0 \subset B$*,* $X_0 \subset X$*, and* $Y_0 \subset Y$*, is the input-output control and communications law,* $\mathcal{CC}_1 \otimes \mathcal{CC}_2$*, with sets*

$$(\mathcal{CC}_1 \otimes \mathcal{CC}_2)[L] = \mathcal{CC}_1 L \times \mathcal{CC}_2 L, \qquad (\mathcal{CC}_1 \otimes \mathcal{CC}_2)[W_{out}] = Z \times C,$$

$$(\mathcal{CC}_1 \otimes \mathcal{CC}_2)[W] = \mathcal{CC}_1 W \times \mathcal{CC}_2 W \times Y \times B, \quad (\mathcal{CC}_1 \otimes \mathcal{CC}_2)[W_{in}] = X \times A,$$

$$(\mathcal{CC}_1 \otimes \mathcal{CC}_2)[W_0] = \mathcal{CC}_1 W_0 \times \mathcal{CC}_2 W_0 \times Y_0 \times B_0, \; (\mathcal{CC}_1 \otimes \mathcal{CC}_2)[W_{in0}] = X_0 \times A_0,$$

and functions

$$\begin{aligned}
\mathrm{msg}(x, w, w_{in}) &= (\mathcal{CC}_1 \mathrm{msg}(x, \mathcal{CC}_1 w, \mathcal{CC}_1 w_{in}), \mathcal{CC}_2 \mathrm{msg}(x, \mathcal{CC}_2 w, \mathcal{CC}_2 w_{in})), \\
\mathrm{stf}_{\mathrm{io-slf}}(w, w_{in}, l) &= (\mathcal{CC}_1 \mathrm{stf}_{\mathrm{io-slf}}(\mathcal{CC}_1 w, \mathcal{CC}_1 w_{in}, \mathcal{CC}_1 l), \\
&\qquad \mathcal{CC}_2 \mathrm{stf}_{\mathrm{io-slf}}(\mathcal{CC}_2 w, \mathcal{CC}_2 w_{in}, \mathcal{CC}_1 l), \pi_B(\mathcal{CC}_1 \mathrm{stf}_{\text{io-out}}(\mathcal{CC}_1 w, \mathcal{CC}_1 w_{in})), \\
&\qquad\qquad \pi_Y(\mathcal{CC}_2 \mathrm{stf}_{\text{io-out}}(\mathcal{CC}_2 w, \mathcal{CC}_2 w_{in}))), \\
\mathrm{stf}_{\text{io-out}}(w, w_{in}, l) &= (\pi_Z(\mathcal{CC}_2 \mathrm{stf}_{\text{io-out}}(\mathcal{CC}_2 w, \mathcal{CC}_2 w_{in}), \mathcal{CC}_2 l), \\
&\qquad\qquad \pi_C(\mathcal{CC}_1 \mathrm{stf}_{\text{io-out}}(\mathcal{CC}_1 w, \mathcal{CC}_1 w_{in}), \mathcal{CC}_1 l)), \\
\mathrm{ctl}(x_{t_\ell}, x, w^{[i]}, {w_{in}}^{[i]}) &= \mathcal{CC}_1 \mathrm{ctl}(x_{t_\ell}, x, \mathcal{CC}_1 w, \mathcal{CC}_1 w_{in}) + \mathcal{CC}_2 \mathrm{ctl}(x_{t_\ell}, x, \mathcal{CC}_2 w, \mathcal{CC}_2 w_{in}).
\end{aligned}$$

2.3 The graph Laplacian and its spectrum

An (undirected) graph $G = (V, \mathcal{E})$ consists of a vertex set V and an edge set $\mathcal{E} \subset V \times V$ of unordered pairs of vertexes, i.e., $(i, j) \in \mathcal{E}$ implies that $(j, i) \in \mathcal{E}$. A weighted graph is an undirected graph where each edge $(i, j) \in \mathcal{E}$ has an associated weight $w_{i,j} \in \mathbb{R}_{\geq 0}$. For a weighted graph $G = (V, \mathcal{E})$, the (weighted) adjacency $A(G) \in \mathbb{R}^{n \times n}$ and the Laplacian $L(G) \in \mathbb{R}^{n \times n}$ are given by

$$A(G)_{i,j} = w_{i,j}$$

$$L(G)_{i,j} = \begin{cases} \sum_{k \neq i} w_{i,k} & i = j, \\ -w_{i,j} & i \neq j. \end{cases}$$

When the specific graph is clear from the context, we simply use A and L. Note that both matrices are symmetric. For convenience, we denote by $\Lambda : \mathrm{Sym}(n) \to \mathrm{Sym}(n)$ the linear map that transforms an adjacency matrix A into the Laplacian L defined by

$$\Lambda(A) = \mathrm{diag}(A\mathbf{1}) - A = L.$$

Properties of the Laplacian matrix include [19]: the vector $\mathbf{1} \in \mathbb{R}^n$ is an eigenvector with eigenvalue 0; $L(G)$ is positive semidefinite; and the dimensionality of the null space of $L(G)$ is equal to the number of connected components of G. As a consequence of these properties, an undirected graph is connected if and only if the second smallest eigenvalue of its Laplacian is greater than zero. Another convenient property of Laplacians is that adding weight to an edge is guaranteed not to decrease any of its eigenvalues [66].

2.4 Proximity graphs and proximity functions

We use proximity graphs as an abstraction of network connectivity among spatially distributed robots. A proximity graph is an association of a set of positions with a weighted graph. Let $\mathcal{P} = (p_1, \ldots, p_n) \in (\mathbb{R}^d)^n$ be a vector of n robot positions, where each robot evolves in $\mathbb{R}^d$. Let $\mathbb{G}(n)$ be the set of weighted graphs whose vertex set is the set of integers between 1 and n (denoted by $\{\{1, \ldots, n\}\}$). Then, we have the following definition [24, 13].

Definition 2.4.1. *A* proximity graph $\mathcal{G} : (\mathbb{R}^d)^n \to \mathbb{G}(n)$ *associates to* $\mathcal{P} \in (\mathbb{R}^d)^n$ *a graph with vertex set* $\{1, \ldots, n\}$*, edge set* $\mathcal{E}_\mathcal{G}(\mathcal{P})$*, where* $\mathcal{E}_\mathcal{G} : (\mathbb{R}^d)^n \to \{1, \ldots, n\} \times \{1, \ldots, n\}$*, and weights* $w_{i,j} \in \mathbb{R}_{>0}$ *for all* $(i, j) \in \mathcal{E}_\mathcal{G}(x)$*. A proximity graph must satisfy that* $\mathcal{G}(p_{\sigma(1)}, \ldots, p_{\sigma(n)})$ *is isomorphic to* $\mathcal{G}(p_1, \ldots, p_n)$ *for any* n*-permutation* σ *and* $(p_1, \ldots, p_n) \in (\mathbb{R}^d)^n$*.* •

For a given proximity graph, we often use the associated *proximity function* $(\mathbb{R}^d)^n \to \mathrm{Sym}(n)$ that maps a tuple $\mathcal{P} \in (\mathbb{R}^d)^n$ to the adjacency matrix $A(\mathcal{G}(\mathcal{P})) \in \mathrm{Sym}(n)$. Note that a proximity graph can be alternatively defined by specifying a proximity function.

Remark 2.4.2. *Examples of proximity functions include the following:*

(i) the r*-disk proximity function,*

$$f_{r\text{-}disk}(p_1, \ldots, p_n)_{i,j} = \begin{cases} 1, & \|p_i - p_j\| \leq r, \\ 0, & otherwise, \end{cases}$$

(ii) the exponentially-decaying proximity function,

$$f_{exp}(p_1, \ldots, p_n)_{i,j} = \exp(-\|p_i - p_j\|),$$

(iii) the approximate r-disk graph, for a sharpness value $k \in \mathbb{R}$,

$$f_{r\text{-}disk\text{-}cont}(p_1, \ldots, p_n)_{i,j} = \frac{1}{1 + \exp(k(\|p_i - p_j\| - r))}.$$

(iv) the spline graph, with $0 < r_{min} < r_{max}$, an approximation of $f_{r\text{-}disk}$, with $r \in [r_{min}, r_{max}]$. The (i,j) entry $f_{spline}(p_1, \ldots, p_n)_{i,j}$ is given by

$$\begin{cases} 0, & \|p_i - p_j\| < r_{min}, \\ 1 - 3(\frac{\|p_i - p_j\| - r_{min}}{r_{max} - r_{min}})^2 + 2(\frac{\|p_i - p_j\| - r_{min}}{r_{max} - r_{min}})^3, & r_{min} \leq \|p_i - p_j\| \leq r_{max}, \\ 1, & s = \|p_i - p_j\| > r_{max}, \end{cases}$$

These examples are particular classes of a larger class of proximity functions defined by $f(p_1, \ldots, p_n)_{i,j} = g_{\text{wgt}}(\|p_i - p_j\|)$, with $g_{\text{wgt}} : \mathbb{R}_{\geq 0} \to \mathbb{R}$. For this paper we consider proximity functions of this form. The algorithm discussed in Chapter 6 requires the added restrictions that g_{wgt} is $\mathcal{C}^2$ and monotonically decreasing, like in examples (ii)-(iv). Some properties of the algorithms proposed in Chapter 6 will also require that the second derivative of g_{wgt} is bounded and g_{wgt} has zero derivative at 0, like in example (iv).

2.5 Elements of nonsmooth analysis

It is possible to define a notion of gradient for locally Lipschitz functions [11]. Let $f : \mathbb{R}^d \to \mathbb{R}$ be locally Lipschitz at $x \in \mathbb{R}^d$. For any $v \in \mathbb{R}^d$, the *generalized directional derivative of f at x in the*

direction v, denoted $f^\circ(x;v)$, is

$$f^\circ(x;v) = \limsup_{y \to x, t \downarrow 0} \frac{f(y+tv) - f(y)}{t}.$$

In contrast, the *one-sided directional derivative, of* f *at* x *in the direction* v, denoted $f'(x;v)$, is

$$f'(x;v) = \lim_{t \to 0} \frac{f(y+tv) - f(y)}{t}.$$

The generalized directional derivative has the property of always being well-defined, whereas the one-sided directional derivative might not exist in some cases. The *generalized gradient of* f at $x \in X$, denoted $\partial f(x)$, is the subset

$$\partial f(x) = \{\xi \in X \ : \ f^\circ(x;v) \geq \xi^T v \text{ for all } v \text{ in } X\}.$$

If f is continuously differentiable at x, then $\partial f(x) = \{\nabla f(x)\}$.

2.6 Nonsmooth analysis of the algebraic connectivity function

Here we specify our scalar measure of network connectivity. Denote the (not necessarily distinct) eigenvalues of $M \in \mathrm{Sym}(n)$ by $\lambda_1(M) \leq \lambda_2(M) \leq \cdots \leq \lambda_n(M)$. We denote by $f_{\lambda_i} : \mathrm{Sym}(n) \to \mathbb{R}$ the function that maps the matrix M to $\lambda_i(M)$. Given a proximity function $f : (\mathbb{R}^d)^n \to \mathrm{Sym}(n)$, we let

$$f_{i-\mathrm{conn}} = f_{\lambda_i} \circ \Lambda \circ f : (\mathbb{R}^d)^n \to \mathbb{R}. \tag{2.1}$$

We refer to $f_{2-\text{conn}}$ as the *algebraic connectivity function.*

Next, we analyze the smoothness properties of the functions $f_{i-\text{conn}}$, for $i \in \{1, \ldots, n\}$. We are particularly interested in $f_{2-\text{conn}}$, but the same results are valid for any $f_{i-\text{conn}}$, and therefore we present them in general. We start by establishing that each f_{λ_i} is globally Lipschitz.

Lemma 2.6.1. *For $i \in \{1, \ldots, n\}$, the function f_{λ_i} is globally Lipschitz with Lipschitz constant* 1. •

Since the composition of Lipschitz functions is also Lipschitz, we have the following corollary.

Corollary 2.6.2. *For $i \in \{1, \ldots, n\}$ and a locally Lipschitz proximity function f, the connectivity function f_{i-conn} is also locally Lipschitz.*

The following result [30] gives gradients of functions f_{λ_i}.

Theorem 2.6.3. *For $i \in \{1, \ldots, n\}$, the generalized directional derivative (in the direction $X \in Sym(n)$) and the generalized gradient of f_{λ_i} at $M \in Sym(n)$ are given by*

$$f^{\circ}_{\lambda_i}(M; X) = \max_{\{v \in \mathbb{S}^n \,:\, Mv = \lambda_i v\}} vv^T \bullet X,$$

$$\partial f_{\lambda_i}(M) = co_{\{v \in \mathbb{S}^n \,:\, Mv = \lambda_i v\}}\{vv^T\}.$$ •

The next result is a consequence of (2.1) and the nonsmooth chain rule [11, Theorem 2.3.10].

Theorem 2.6.4. *Given a continuously differentiable proximity function, $f : (\mathbb{R}^d)^n \to Sym(n)$, we have at $\mathcal{P} \in (\mathbb{R}^d)^n$, and $L = \Lambda(f(\mathcal{P}))$,*

$$\partial f_{i-conn}(\mathcal{P}) \subseteq (\text{vec}(\partial f_{\lambda_i}(L)))^T (\nabla \text{vec}(L)).$$ •

2.7 A taxonomy of connectivity problems

Given our current notions of connectivity, we can pose a series of problems.

- Given n robots communicating over a proximity graph, how do we move the robots so as to maximize f_{conn}, subject to some position constraints?
- Given n robots and some control objective, f, how do we move the robots so as to maximize f, subject to the constraint that the communication graph remains connected?
- Given n robots, a connectivity threshold c and some control objective, f, how do we move the robots so as to maximize f subject to f_{conn} never going below c?

Kim and Mesbahi [26] and solve problem 1, although none of the existing solutions work in a distributed manner. Zavlanos and Pappas [68] solution to 2 is not distributed, but allows for a general range of motion, which we would like out of a distributed algorithm for 2. Notarstefano et. al. [41] solve problem 2 in a distributed way, but for fixed network topology. De Gennaro et. al. [16] provide a distributed solution to problem 3, but the communication complexity of their solution exceeds that of exchanging the full state of the system and running a centralized algorithm.

Chapter 3

Structure of this work

There are two main thrusts to our research.

Part III discusses the primary direction of our work. Our recent research fits almost entirely into this chapter. In it, we focus on proximity graphs induced by inter-robot distances (see the discussion on *proximity functions* in Section 2.4). Given proximity graphs from this family, and a measure of connectivity we seek to maintain, we create distributed algorithms to control the motions of individual robots to make guarantees about the appropriate measure of connectivity. We develop two approaches of this form, and compare them with one another and with the established research in the field. Section 6.5.1 of Chapter 6 compares one of our algorithms with closely related algorithms from the literature.

Part II introduces an earlier piece of research, discussing connectivity in a radically different setting. We hope the juxtaposition of this work with the two approaches discussed in Part III serves to provide a broader picture of multi-robot coordination and control of spatially-induced connectivity. In Part II we develop connectivity algorithms based on the assumptions used in [45] to develop similar algorithms. The motivation of this set of assumptions was partly due to an actual spacecraft design

being proposed at the Jet Propulsion Laboratory. The setup differs from that in Part III in the following ways

(i) Robots live in 3 dimensions

(ii) Robots have directional sensors.

  (a) Edges exist between any two robots with line-of-sight between their directional sensors.

  (b) Symbolic information can be communicated between any pair of robots at any time

  (c) Relative position information can only be exchanged over edges in the communication graph

  (d) Robots control directionality of their sensors. Robot position follows dynamics induced by uncontrolled drift.

(iii) We are concerned with the connectivity of the union of all graphs induced by robot positions and orientations over a finite time window. Unlike our approach in Part III, we are not concerned about connectivity of the communication graph at any particular instant in time.

In Part IV we attempt to summarize our findings in a unified discussion of connectivity among coordinated robotic agents.

Part II

Connectivity of union of line of sight graphs

Chapter 4

Weak spacecraft connectivity

4.1 Formation initialization work

Deploying large structures in space requires multiple spacecraft to coordinate their activities, due, in part, to the limited payload capabilities of launch vehicles. One application that requires such coordination is the deployment of large-baseline interferometers for science imaging missions. Key aspects of spacecraft coordination which are likely to be used in a broad variety of contexts include: (i) formation initialization, i.e., the establishment and maintenance of relative dynamic state information and/or on-board inter-spacecraft communication; (ii) formation acquisition, i.e., making the group of spacecraft attain a desired geometry; and (iii) formation regulation and tracking, i.e., maintaining fixed inter-spacecraft range, bearing, and inertial attitudes with high accuracy along the execution of a desired trajectory.

In this Section, we focus our attention on the formation initialization problem. This problem is especially important for spacecraft operating in deep space, where conventional Earth-based GPS does not provide sufficiently accurate position information. Here, we consider a spacecraft model motivated, in part, by the design possibilities of NASA's "Terrestrial Planet Finder" mission. Our

spacecraft model is similar to the one proposed in [45]. The spacecraft have laser-based directional relative position sensors, like the kind described in [20], which require two sensors to lock on to each other before getting a position measurement. Each of the spacecraft has a sun-shield which must be oriented so as to protect sensitive astronomical instruments from solar radiation. The spacecraft are assumed to be in deep space, far from the effects of gravitational curvature.

4.1.1 Preliminaries

Each spacecraft consists of a rigid body containing instruments on one side, which need to be shielded from the sun (see Fig. 4.1). To serve this purpose, a *sun shield* is mounted to the spacecraft

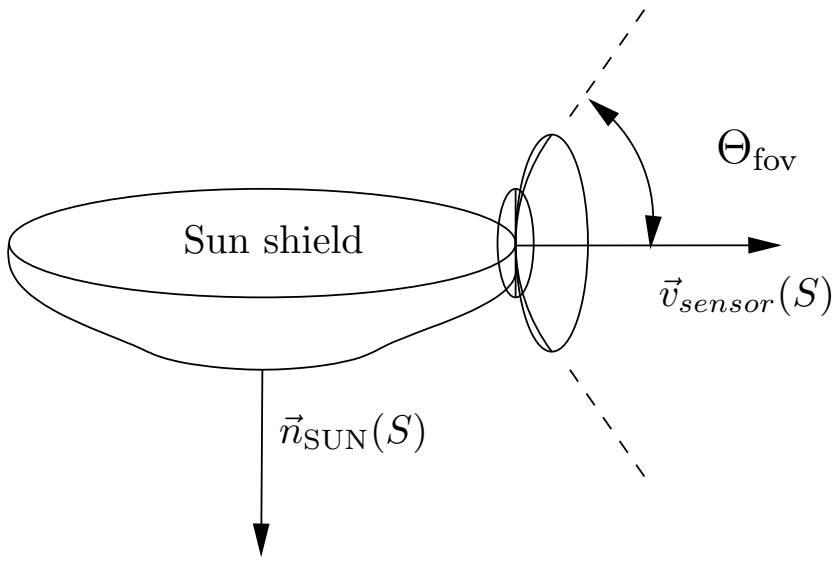


Figure 4.1: Configuration of spacecraft geometry, and body frame definition, as in [20, 45].

body on the side opposite the instruments. The *sun shield normal vector*, $\vec{n}_{\text{SUN}}(S)$, indicates the direction of the sun shield of spacecraft S. We make the approximation that the sun is an infinite distance away, and therefore the vector to the sun, $\vec{v}_{\text{SUN}}$, is the same for each feasible spacecraft position. In order to operate without damaging the instrumentation, each spacecraft must maintain the constraint $\vec{n}_{\text{SUN}}(S) \cdot \vec{v}_{\text{SUN}} \geq \cos(\Theta_{\text{sun}})$ for some pre-specified Θ_{sun} at all times. Relative position

and velocity measurements between two spacecraft are made through the *metrology sensors* of the two craft. The metrology sensor of spacecraft S senses within a conical region (C_S) with a half angle of Θ_{fov} (assumed here to be greater then $\frac{\pi}{4}$ unless otherwise stated). The *sensor cone centerline* of S is an infinite ray down the axis of rotational symmetry of the sensor cone defined by the unit vector $\vec{v}_{\text{SENSOR}}(S)$. Accurate orientation information is available for all spacecraft through measurements of reference stars. Thus we only have to worry about obtaining relative position and velocity information for each spacecraft. The spacecraft are placed such that the curvature of earth's gravitational field has a negligible effect (for instance a Lagrange point). We therefore assume that if no spacecraft undergo translational acceleration, then the spacecraft move with constant (initially unknown) velocity in straight lines relative to each other.

Definition 4.1.1. *The* global frame of reference *is an arbitrary orthonormal frame,* $GF = \{X_g, Y_g, Z_g\}$, *where* $X_g = \vec{v}_{SUN}$. *For a spacecraft* S, P_S *denotes the position of the center of mass of* S *in the frame* GF. *The* center of mass frame *of* S, $CMF(S)$, *corresponds to translating the global frame* GF *to* P_S. *The* body frame *of* S, $BF(S) = \{\hat{X}_S, \hat{Y}_S, \hat{Z}_S\}$ *is defined by* $\hat{X}_S = \vec{n}_{SUN}(S)$, $\hat{Z}_S = \vec{v}_{SENSOR}(S)$ *and* $\hat{Y}_S = \hat{Z}_S \times \hat{X}_S$. *In this frame,* $\{0, 0, 0\}$ *is at the center of mass of* S.

The state of each spacecraft $S \in \{S_1, \cdots, S_n\}$ can be described by $(P_S, M_S) \in \mathbb{R}^3 \times SO(3)$.$M_S$ transforms $BF(S)$ onto $CMF(S)$ and P_S defines the translation between $CMF(S)$ and GF. The spacecraft are fully actuated.

The sensor cone $C_S : \mathbb{R}^3 \times SO(3) \to 2^{\mathbb{R}^3}$ of S is

$$C_S(P_S, M_S) = \{\vec{x} \in \mathbb{R}^3 \ : \ \frac{[0,0,1]^T M_S(\vec{x} - P_S)}{\|\vec{x} - P_S\|} \leq \cos(\Theta_{\text{fov}})\}. \tag{4.1}$$

When it is clear from the context, we will use the simpler notation C_S. In order to get a relative position

reading between two spacecraft, S_1 and S_2, S_1's metrology sensor must point at S_2. This condition is called *sensor lock*. Formally, two spacecraft, S_1 and S_2, achieve **sensor lock** if and only if $P_{S_1} \in C_{S_2}$ and $P_{S_2} \in C_{S_1}$.

The algorithms we present all require the n spacecraft performing the algorithm to be split into two disjoint groups, G_1 and G_2 such that $G_1 \cup G_2 = \{S_1, ..., S_n\}$. This can either be done a priori before launch or (preferably) with a distributed algorithm prior to running formation initialization (see [33]).

Two spacecraft, S_1 and S_2, are said to be maintaining the **Opposing Sensor Constraint** if $\vec{v}_{\text{SENSOR}}(S_1) = -\vec{v}_{\text{SENSOR}}(S_2)$.While this constraint is not strictly necessary for a correct solution to the formation initialization problem, we will show in Section 4.1.7 that it is a convenient constraint to work with. Note that this does not fully constrain the relative orientation of S_1 with respect to S_2. When specifying an algorithm requiring the **Opposing Sensor Constraint**, we will often specify the more restrictive constraint

$$M_{S_1}^{-1} M_{S_2} = M_{\text{opp}} = \text{diag}(1, -1, -1).$$

We call this constraint the **Opposing Frame Constraint**. In addition to maintaining the **Opposing Sensor Constraint**, the **Opposing Frame Constraint** also guarantees that if spacecraft S_1 verifies the sun-angle constraint, then S_2 also verifies it.

4.1.2 Algorithm definition

Our definition of "algorithm" here differs in some ways from the one we use elsewhere in this paper. In what follows, D_{msg} denotes the set of possible messages a spacecraft can communicate at any instant, and $D_{\text{sensor}} = (\mathbf{Z}_2 \times \mathbb{R}^3 \times \mathbb{R}^3)^n$ the set of possible sensor cone readings for a spacecraft.

Definition 4.1.2 (Algorithm notion)**.** *An algorithm is a tuple* $A_S = (STATE_{S,0}, F_S, G_S, \delta t_{step})$*, where* $STATE_{S,0} \in D_{STATE}$*, the initial internal state of* S*, contains no information about the location of the other spacecraft and* F_S *is a map of the form*

$$F_S : \mathbb{R} \times SO(3) \times D_{STATE} \to \mathbb{R}^3$$

$$(t, M_S, STATE_S) \mapsto \omega_S,$$

and G_S *is a map of the form*

$$G_S : \mathbb{R} \times SO(3) \times D_{STATE} \times D_{msg}^{n-1} \times D_{sensor} \to D_{STATE} \times D_{msg}$$

$$(t, M_S, STATE_S, MSG_{S,in}, SENSOR_S) \mapsto (STATE_S, MSG_{S,out}).$$

Definition 4.1.3. (Execution of an algorithm): *An execution by a spacecraft* S *of an algorithm* $A_S = (STATE_S, F_S, G_S, \delta t_{step})$ *during the time interval* $[t_0, t_f]$ *is given by* $t \in [t_0, t_f] \to (P_S(t), M_S(t)) \in \mathbb{R}^3 \times SO(3)$ *and* $STATE_S : [t_0, t_f] \to D_{STATE}$ *defined as follows:*

- $\dot{P}_S(t) = V_S$*, for some constant* $V_S \in \mathbb{R}^3$*;*
- $\dot{M}_S(t) = \widehat{F_S}(t, STATE_S(t))M_S(t)$*,* $t \in [t_0, t_f]$*, where* $\hat{\omega}$ *is the matrix operator for the cross product with* $\omega \in \mathbb{R}^3$*;*
- $STATE_S$ *is the piecewise constant function defined by*

$$STATE_S(t_{i+1}) = G_S((t_i, M_S(t_i), STATE_S(t_i), MSG_{S,in}(t_i), SENSOR_S)(t_i)),$$

for $i = 0, \ldots, m-1$, with $t_0, t_1, \ldots, t_m \in [t_0, t_f]$ a finite increasing sequence, where $t_i = k\,\delta t_{step}$ for some $k \in \mathbb{N}$ or t_i corresponds to the time instant when a change occurs in the value of the sensor cone readings. The initial value of $STATE_S(t_0)$ is $STATE_{S,0}$.

The lack of concrete specification of D_{msg} and D_{STATE} reflects our intent to provide lower bounds on algorithmic performance for spacecraft with a wide range of computational and communication capabilities. In practice, the working algorithms we present in Section 4.1.8 require basic computational capabilities on the part of each spacecraft.

4.1.3 Prior work

A fairly extensive bibliography of missions which plan to use spacecraft formation flying can be found in [64]. These include Terrestrial Planet Finder [28], EO-1 [4], TechSat-21 [15] and Orion-Emerald [21]. A driving motivation behind formation flying research is that of large aperture adaptive optics in space, e.g. [28]. Optical devices such as the ones described in [17] could combine the advantages of multi-mirror adaptive optics with those of space telescopes. A good overview on current research on formation flying for optical missions is contained in [20]. Most of the work on control algorithm design has focused on formation acquisition and tracking. A survey of algorithms is given in [51]. Leader-following approaches, e.g. [25, 39], and virtual structures approaches, e.g. [5], prescribe the overall group behavior by specifying the behavior of a single leading agent, either real or virtual. Motion planning and optimal control problems are analyzed in [22]. The only work known to us that has dealt in detail with formation initialization is [45].

4.1.4 Total angle traversed and solid angle covered

Here, we present the notions of total angle traversed and solid angle covered during the execution of an algorithm.

In 3-D, recall that $M_S = [m_x, m_y, m_z]$ is an orthonormal basis matrix representing the orientation of spacecraft S. From Equation 8.6.5 of [34], we have $\hat{\omega} = \dot{M}_S M_S^{-1}$. , where $\hat{\omega}$ is the matrix operator for "cross product with ω."

The total angle traversed during the execution of an algorithm in 3-D is therefore

$$\int_{t_0}^{t_f} \sqrt{\hat{\omega}_{1,2}^2 + \hat{\omega}_{1,3}^2 + \hat{\omega}_{2,3}^2} dt.$$

One can think of the 2-D problem as the 3-D problem with rotations confined to the $\{Y, Z\}$ plane. Under this constraint, the previous expression reduces to

$$\int_{t_0}^{t_f} |\hat{\omega}_{2,3}| dt.$$

It will be useful to compute the total solid angle covered by the sensor cone C_S of a spacecraft performing a formation initialization algorithm in 3-D. If a spacecraft, S, with sensor cone field of view Θ_{fov} rotates by an angle of π about an axis l where $l \cdot \vec{v}_{\text{SENSOR}}(S) = \cos(\Theta)$, $\Theta > \Theta_{\text{fov}}$, the new solid angle covered in this sweepcan be found by tracing a band about the unit sphere and calculating its area. See Figure 4.2 for an illustration.

Recall that the solid angle of a cap of half angle α is $\int_0^{\alpha} 2\pi \sin(t) dt$. The area of this band can be found by subtracting caps of half angles $\Theta - \Theta_{\text{fov}}$ and $\pi - \Theta - \Theta_{\text{fov}}$ from the unit sphere and

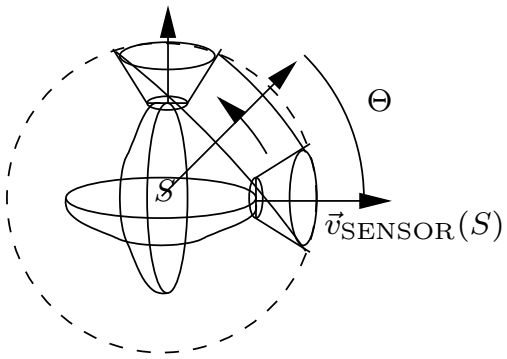


Figure 4.2: Method to compute rate of change of solid angle swept.

dividing by 2. , giving a result of

$$\frac{4\pi - 2\pi(1 - \cos(\pi - \Theta - \Theta_{\text{fov}})) - 2\pi(1 - \cos(\Theta - \Theta_{\text{fov}}))}{2}.$$

Dividing by π gives a rate of change of coverage of solid angle for this operation , when performed at angular velocity ω, as $2\|\omega\| \sin(\Theta) \sin(\Theta_{\text{fov}})$. A similar argument gives the expression for when $\Theta \leq \Theta_{\text{fov}}$ as $\|\omega\|(1 + \sin(\Theta) \sin(\Theta_{\text{fov}}) - \cos(\Theta) \cos(\Theta_{\text{fov}}))$

The total solid angle covered by a spacecraft S executing an algorithm, A, between times t_0 and t is then

$$F_{\text{sld}}(t) = \int_{t_0}^{t} f_{\text{sld}}(\omega(\tau))d\tau,$$

where $f_{\text{sld}}(\omega) : \mathbb{R}^3 \to \mathbb{R}$ is defined by $f_{\text{sld}}(\omega) = 2\|\omega \times \vec{v}_{\text{SENSOR}}(S) \sin(\Theta_{\text{fov}})\|$ for $\arccos(\omega \cdot \vec{v}_{\text{SENSOR}}(S)/\|\omega\|) > \Theta_{\text{fov}}$, and $f_{\text{sld}}(\omega) = \|\omega \times \vec{v}_{\text{SENSOR}}(S) \sin(\Theta_{\text{fov}})\| + \|\omega\| - |\omega \cdot \vec{v}_{\text{SENSOR}}(S)|$ otherwise. For us, the total solid angle covered by S during the course of the algorithm to be $F_{\text{sld}}(t_f) + \alpha_0$ where t_f is the earliest time at which formation initialization is guaranteed to be complete and $\alpha_0 = 2\pi(1 - \cos(\Theta_{\text{fov}})$ is the solid angle contained in $C_S(t_0)$ at time t_0.

Remark 4.1.4. *Note that* $0 \leq f_{sld}(\omega) \leq 2\|\omega\| \sin(\Theta_{fov})$. •

Analogously, the total angle covered by a spacecraft S performing an algorithm A in 2-D between times t_0 and t is

$$F_{\text{angle}}(t) = \int_{t_0}^{t} |\omega| dt.$$

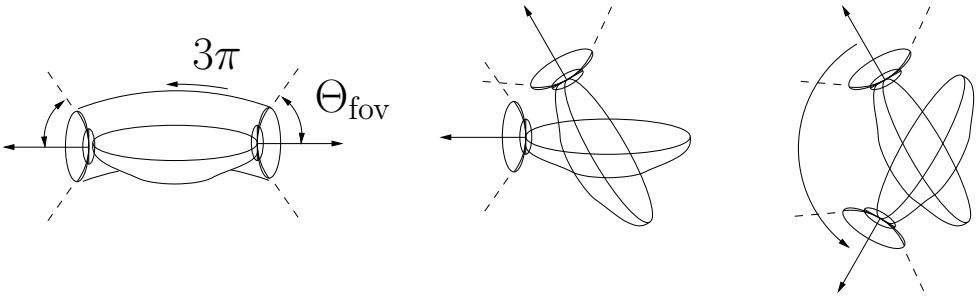
4.1.5 Formation initialization problem

Formation initialization solutions entail establishing communication and/or relative position information. Frequently they also involve moving the spacecraft to an initial formation from which another formation control algorithm can take over. Here we restrict ourselves to the establishment of relative position and velocity information between each pair of spacecraft. We assume that this information can come from any combination of direct sensor readings, odometry and communication with other spacecraft.

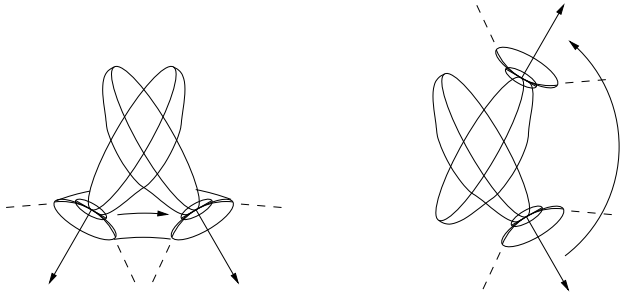
Definition 4.1.5. *Let* $[t_s, t_f]$ *be the duration of time during which a formation initialization algorithm runs. Define* $G(t)$ *to be the* ***relative position connectivity network*** *at time* t*, defined by* $G(T) = (V, E)$ *where* $v(S_i) \in V$ *correspond to the spacecraft* S_i*, and the edge* $(v(S_i), v(S_j))$ *is in* E *if and only if spacecraft* S_i *and* S_j *are in a state of sensor lock. A solution to the formation initialization problem is one that guarantees that the graph* $\cup_{t\in[t_s,t_f]} G(t)$ *is connected, so long as no two spacecraft collide by* t_f.

4.1.6 Details of prior work

The multi-spacecraft algorithm proposed in [45] to solve formation initialization is briefly described in Table 4.1. We discuss its correctness in Theorem 4.1.7.



Step 7 Step 8 Step 9



Step 10 Step 11

Figure 4.3: Stages of the algorithm described in Table 4.1.

Name:	Formation Initialization Algorithm
Assumes:	Spacecraft model in Section 4.1.1.

1: **if** $S_i \in G_1$ **then**
2: Rotate to align M_{S_i} with I_3
3: **else**
4: Rotate to align M_{S_i} with M_{opp}
5: **end if**
6: Wait for common start time t_s
7: Rotate by 3π about X_{S_i}.
8: Rotate $-\Theta_{\text{tilt}}$ (in this case 25 degrees) about Y_{S_i}.
9: Rotate $2\Theta_{\text{tilt}}$ about Y_{S_i}.
10: Rotate by π about X_{S_i}.
11: Rotate $2\Theta_{\text{tilt}}$ about Y_{S_i}.
{This is the end of the rotational component of the algorithm}
12: Rotate $-\Theta_{\text{tilt}}$ about Y_{S_i}.
13: Wait for some time $t_{\text{near field}} > 0$
14: **if** $S_i \in G_1$ **then**
15: Begin translating along Z_{S_i} with speed v_{max}, where v_{max} is the maximum relative velocity between any two craft.
16: **end if**

Table 4.1: Formation Initialization algorithm proposed in [45].

4.1.7 Correctness and optimality of formation initialization algorithms

We start this section by providing a necessary condition for the correctness of any formation initialization algorithm. Then, we proceed to use this condition as the basis for a series of optimality bounds. We also present optimality results which justify the **Opposing Sensor Constraint** and allow us to more easily reason about the n spacecraft case (where $n > 2$).

Theorem 4.1.6. *Let S be executing a correct formation initialization algorithm in d dimensions, with $d \in \{2, 3\}$. For every $v \in \mathbb{R}^d$, let t_v be the first time such that $v \in C_S(t_v) = C_S(P_S(t_v), M_S(t_v))$. Then, there must exist $t^* > t_v$ such that $-v \in C_S(t^*)$.*

Proof. For simplicity, let $\text{vers}(u) = u/\|u\|$, for $u \in \mathbb{R}^d$. Consider two spacecraft, S_1 and S_2. S_2 travels in the plane defined by it's velocity (V_{S_2}), and $p_{\text{cl}}(S_1, S_2)$, where $p_{\text{cl}}(S_1, S_2)$ is the point of closest approach between S_1 and S_2 in $CMF(S_1)$. At time t S_2 makes an angle with $p_{\text{cl}}(S_1, S_2)$ of $\arctan(\frac{\|V_{S_2}\|}{\|p_{\text{cl}}(S_1,S_2)\|}t+t_0)$

for some t_0. S_2's initial conditions can be chosen to match any arbitrary V_{S_2}, $p_{\text{cl}}(S_1, S_2)$ and t_0. Because of this, given an ϵ and times, t_1 and t_2, $\text{vers}(P_{S_2})$ can be made to stay within an angle of ϵ of $-\text{vers}(V_{S_2})$ until time t_1, and move to within an angle of ϵ of $\text{vers}(V_{S_2})$ by t_2. Let t_1 be the first time at which the minimum angle between any ray in $C_{S_1}(t_1)$ and $\text{vers}(-V_{S_2})$ is less then or equal to ϵ and t_2 be the first time at which $C_{S_1}(t_2)$ includes $\text{vers}(-V_{S_2})$. In order to ensure S_1 finds S_2, $C_{S_1}(t^*)$ must include $\text{vers}(V_{S_2})$ at some time $t^* > t_1$. Since ϵ was picked arbitrarily and the sensor cone is always closed, $C_{S_1}(t^*)$ must include $\text{vers}(V_{S_2})$ at some time $t^* > t_2$. □

Theorem 4.1.7. *The algorithm stages described in Steps 1-12 of Table 4.1 are not, by themselves, sufficient to solve the formation initialization problem.*

Proof. Let $S \in G_1$ perform this algorithm. By Theorem 4.1.6, for any vector v, $C_S(t)$ must contain $-v$ at least once before the last time $C_S(t)$ contains v. But each $v \in R_{down}$ is last in $C_S(t)$ during Step 9, and no $v \in \{u \in CMF(S) \ : \ -u \in R_{down}\}$ is in $C_S(t)$ before Step 10. Thus $R_{down}(S)$ does not satisfy this condition. □

For our purposes, we will consider the algorithm which minimizes the maximum worst-case total angle traversed of any spacecraft S_i to be the optimal algorithm. Other reasonable options would include the algorithm which minimizes the worst-case sum over all spacecraft S_i of the total angle traversed.

Let is now justify that **Opposing Sensor Constraint** is optimal.

Theorem 4.1.8. *(Justification of the **Opposing Sensor Constraint**): Let S_1 and S_2 be two spacecraft. The most optimal algorithm to guarantee that S_1 and S_2 attain sensor lock is one which uses the **Opposing Sensor Constraint**.*

Proof. Imagine there is some algorithm A which achieves sensor lock between S_1 and S_2 in time t_{lock}. Create a new algorithm A^* in which S_1 implements A, but S_2 maintains the **Opposing Sensor Constraint** with S_1. If S_2 had been following A, the apex of $C_{S_2}(t_{lock})$ would be in $C_{S_1}(t_{lock})$ at time t_{lock}. Since S_1 is following A in algorithm A^*, the apex of $C_{S_2}(t_{lock})$ is in $C_{S_1}(t_{lock})$ when both craft follow A^*. By symmetry properties of the **Opposing Sensor Constraint**, the apex of $C_{S_1}(t_{lock})$ is in $C_{S_2}(t_{lock})$, thus guaranteeing sensor lock at or before time t_{lock}. This means that for any algorithm, A, which guarantees sensor lock, a modified algorithm (A^*) which maintains the **Opposing Sensor Constraint** can be constructed such that A^* guarantees sensor lock in at most as much worst-case rotation as A. □

The next result shows an equivalence between worst-case bounds for 2 spacecraft and worst-case bounds for any number $n > 2$ of spacecraft.

Theorem 4.1.9 (Extending worst-cases to n Spacecraft)**.** *Given a spacecraft S_n with sensor cone half-angle Θ_{fov}, and any $\epsilon > 0$, the worst-case total angle traversed by S_n while performing a correct algorithm with $n-1$ other spacecraft is identical to the worst-case total angle traversed by a spacecraft with sensor cone half-angle $\Theta_{fov} + \epsilon$ performing a correct algorithm with one other spacecraft.*

Proof. Let t_{worst} be the worst-case time for 2 spacecraft to find each other given a maximum angular velocity of ω_{max}. Clearly the worst-case time for n craft is no worse then this. Pick the initial conditions of the first $n-1$ spacecraft arbitrarily. Let C be the set of communications the first $n-1$ craft would send if they start from these conditions and fail to achieve sensor lock with S_n by time t_{worst}. Let T be the trajectory S_n would take given communications C. Let A_t be the algorithm for two spacecraft, S_1 and S_2, under which each S_1 blindly follows T and S_2 maintains the opposing sensor constraint with respect to S_1. Let P_{worst} and v_{worst} be the initial position and velocity of S_1 with respect to S_2 that achieves the worst-case total angle traversed for S_1 under A_t. In the n spacecraft case, pick some

spacecraft S_i. Set the initial position and velocity of S_n with respect to S_i to be λP_{worst} and λv_{worst} for λ such that $\min_{t\in[0,t_{\text{worst}}]}(\|P_{\text{worst}} + v_{\text{worst}}t\|)\lambda > \frac{r_{\text{worst}}}{\sin(\epsilon)}$. Since $S_1, \cdots, S_{n-1}$ never get more then r_{worst} apart, these spacecraft are contained within a ball of radius r_{worst} centered at S_i. By construction of λ, these craft stay within an angular ball of ϵ from S_n's point of view, and thus none of these craft achieve sensor lock with S_n before time t_{worst}. □

Theorem 4.1.9 allows the result from Theorem 4.1.8 to be generalized to any number of spacecraft. In addition, we will use Theorem 4.1.9 throughout the remainder of the paper to allow us to analyze worst-case total angle bounds by considering the 2 spacecraft case.

4.1.8 Provably correct formation initialization algorithms

Having given lower bounds on what is necessary for a correct formation initialization solution, in this section we set out to answer whether the problem as we pose it has a solution. Section 4.1.9 describes an algorithm from the literature for a 2-D variant of this problem. Section 4.1.10 presents a purely rotational algorithm for formation initialization in 3-D and Theorem 4.1.14 gives a proof of its correctness. Section 4.1.11 provides an algorithm which comes closer to the optimality bounds presented in Section 4.1.7 at the expense of other practical considerations. This algorithm is presented as a demonstration of the tightness of the optimality bounds.

4.1.9 Formation initialization in two dimensions

To prove the correctness of the algorithm in 3-D, we will need a simpler algorithm for the 2-D case, which we term "in-plane search". This algorithm, described in Table 4.2, solves the formation initialization problem for a group of spacecraft residing in a plane, see [45]

Proposition 4.1.10 ([45]). *With the spacecraft model in Section 4.1.1, the* PLANAR SPACECRAFT

Name:	PLANAR SPACECRAFT LOCALIZATION ALGORITHM
Goal:	Solve the Formation Initialization problem in 2-D
Assumes:	Spacecraft model in Section 4.1.1

1: **if** $S_i \in G_1$ **then**
2: Turn to common reference orientation Θ_{start}
3: **else**
4: Turn to $\Theta_{\text{start}} + \pi$
5: **end if**
6: At synchronized start time t_s, begin rotating with constant angular velocity $\omega > 0$. Continue this rotation for 3π radians.

Table 4.2: The PLANAR SPACECRAFT LOCALIZATION ALGORITHM.

LOCALIZATION ALGORITHM *achieves formation initialization.*

Note that the PLANAR SPACECRAFT LOCALIZATION ALGORITHM achieves the lower bound from Theorem 4.1.6.

4.1.10 Spatial spacecraft localization algorithm

Both the description of the full 3-D algorithm and its proof of correctness require some additional specific definitions, that we briefly expose next.

For the purpose of this algorithm, we will define $\Theta_{\text{tilt}} = \min\{\Theta_{\text{sun}}, \Theta_{\text{fov}}\}$ and assume $\Theta_{\text{fov}} \geq \frac{\pi}{4}$.

Definition 4.1.11. *Let S be a spacecraft. Define*

- $R_1(S) = \{\vec{u} \in CMF(S) \ : \ \vec{u} \cdot X_S \leq 0\}$;
- $R_2(S) = CMF(S) \setminus R_1(S)$.

Remark 4.1.12. *Let Θ_{tilt} be an angle such that $\frac{\pi}{2} - \Theta_{fov} < \Theta_{tilt} < \Theta_{fov}$. $R_1(S)$ is chosen so as to be included within the region swept out by spacecraft S's sensor cone while it is tilted by an angle Θ_{tilt} towards the sun axis and performing a 3π rotation about the sun axis. $R_2(S)$ is chosen so as to be included within the region swept out by spacecraft S's sensor cone while it is tilted $\frac{\pi}{2} - \Theta_{fov} < \Theta_{tilt} < \Theta_{fov}$*

away from the sun axis and performing a 3π *rotation about the sun axis. Also, note that in the frame* $CMF(S)$, $R_1(S) \cup R_2(S) = \mathbb{R}^3$. •

The full 3-D algorithm will invoke the subroutine described in Table 4.3.

Name:	3-D REGION SWEEP ALGORITHM
Goal:	Scan a region for use as a subroutine by SPATIAL SPACECRAFT LOCALIZATION ALGORITHM
Inputs:	(i) A spacecraft, S_i (ii) An integer, $n \in \{1, 2\}$, indicating the region to be swept
Assumes:	(i) Spacecraft model in Section 4.1.1. (ii) $\Theta_{\text{fov}} \geq \frac{\pi}{4}$ and $\Theta_{\text{fov}} + \Theta_{\text{sun}} \geq \frac{\pi}{2}$.

Require: At the start of this subroutine, there exist matrices $M_1, M_2 \in SO(3)$ such that for all $S_i \in G_1$, $M_{S_i} = M_1$, for all $S_j \in G_2$, $M_{S_j} = M_2$, $M_1[1,0,0]^T = M_2[1,0,0]^T$ and $M_1[0,0,1]^T = -M_2[0,0,1]^T$.

Require: At the start of this subroutine, $[0,0,1]M_1[0,1,0]^T = 0$.

1: Set $\Theta_{\text{rot}} = [0,0,1]M_S[0,0,1]^T(-1^n) \cdot \Theta_{\text{tilt}}$
2: Rotate by Θ_{rot} about Y_{S_i}
3: Begin rotating about X_{S_i} by a constant angular velocity ω. Continue this rotation for 3π radians and then stop.
4: Rotate by Θ_{rot} about Y_{S_i}

Table 4.3: The 3-D REGION SWEEP ALGORITHM.

At the end of the execution of the 3-D REGION SWEEP ALGORITHM, if S_i is in G_1, then $R_n(S_i)$ has been swept, otherwise S_i has maintained an orientation such that for all S_j in G_1 $M_{S_i}[0,0,1]^T = -M_{S_j}[0,0,1]^T$. With these ingredients, we can now define the SPATIAL SPACECRAFT LOCALIZATION ALGORITHM in Table 4.4.

Let us discuss the correctness of the SPATIAL SPACECRAFT LOCALIZATION ALGORITHM. As in Section 4.1.9, we reduce the problem to that of two spacecraft finding each other. Call these spacecraft $S_1 \in G_1$ and $S_2 \in G_2$. Recall that S_2's motion in $CMF(S_1)$ is along a straight line with constant velocity. Consider then the two half-spaces defined by the $\{Y, Z\}$ plane in $CMF(S_1)$. Because S_2 moves with constant velocity with respect to S_1, it can cross from one half-space to the other at most once. The paths it can take are as follows. S_2 can begin in $R_1(S_1)$ and cross to $R_2(S_1)$ at most once.

Name:	Spatial spacecraft localization algorithm
Goal:	Solve the Formation Initialization problem in 3-D
Assumes:	(i) Spacecraft model in Section 4.1.1. (ii) $\Theta_{\text{fov}} \geq \frac{\pi}{4}$ and $\Theta_{\text{fov}} + \Theta_{\text{sun}} \geq \frac{\pi}{2}$.

1: **if** $S_i \in G_1$ **then**
2: Rotate to align M_{S_i} with I_3
3: **else**
4: Rotate to align M_{S_i} with M_{opp}
5: **end if**
6: Wait for common start time t_s
7: Call **3-D region sweep algorithm** on S_i and $R_1(S_i)$
8: Call **3-D region sweep algorithm** on S_i and $R_2(S_i)$
9: Call **3-D region sweep algorithm** on S_i and $R_1(S_i)$

Table 4.4: The Spatial spacecraft localization algorithm.

Likewise S_2 can begin in $R_2(S_1)$ and cross into $R_1(S_1)$ at most once.

Because we make no assumptions about the speed at which these spacecraft take these paths, or at which part of the path they start, handling these cases will automatically handle the cases for paths that fail to cross the $\{Y, Z\}$ plane.

Lemma 4.1.13 (Partial reduction to in-plane search)**.** *Doing a 3π sweep (turning about the sun line) through $R_n(S)$, $n \in \{1, 2\}$, $S \in G_1$, finds all spacecraft in G_2 that stay in $R_n(S)$ during the entire duration of the 3π rotation.*

Proof. Projecting the centerline of the cone and the spacecraft path onto the $\{Y, Z\}$ plane in $CMF(S)$ reduces this to the 2-D algorithm. In the cases where $R_n(S)$ contains points which project directly onto $(0, 0)$ there can be a collision in the 2-D projection which does not correspond to a collision of the craft in 3-D. In these cases, the sensor cone of S_1 always contains all such points, and any colliding craft are found. □

Finally, we are in a position to establish the correctness of the full 3-D algorithm.

Theorem 4.1.14. *With the spacecraft model in Section 4.1.1, the* SPATIAL SPACECRAFT LOCALIZATION ALGORITHM *solves the formation initialization problem.*

Proof. Consider two spacecraft, S_1 and S_2. Let S_2 start in $R_{begin}(S_1)$ and end in $R_{end}(S_1)$. If $R_{begin}(S_1) = R_{end}(S_1)$ we are done. Otherwise S_1 must scan $R_{end}(S_1)$ at least once after the first scan of $R_{begin}(S_1)$. If the scan of $R_{begin}(S_1)$ did not find S_2, then S_2 must be in $R_{end}(S_1)$

If S_2 never crosses the $\{Y, Z\}$ plane, either the scan of $R_1(S_1)$ or the scan of $R_2(S_1)$ must find it. Otherwise, S_2 starts in one region and ends in the other. The sequence of region sweeps performed by S_1 guarantee that S_1 will scan the region S_2 starts in at least once before scanning the region S_2 ends in. If S_2 is not found when S_1 first performs a sweep of the region in which S_2 begins (call this $R_{begin}(S_1)$), then S_2 must be in the remaining region ($R_{end}(S_1)$) by the end of the sweep. Since this was the first sweep of $R_{begin}(S_1)$, S_1 must scan at $R_{end}(S_1)$ at least once after this point and find S_2. □

Remark 4.1.15. *The* SPATIAL SPACECRAFT LOCALIZATION ALGORITHM *sweeps a total solid angle of* $9\pi + \frac{5\Theta_{tilt}}{\sin\Theta_{fov}}$ *and performs rotations totaling* $9\pi + 5\Theta_{tilt}$*, where* $\Theta_{tilt} = \min\left(\frac{\pi}{2} - \Theta_{fov}, \Theta_{sun}\right)$. •

4.1.11 Wait and check algorithm

As pointed out in Remark 4.1.15, the provably correct SPATIAL SPACECRAFT LOCALIZATION ALGORITHM is far from optimal both in terms of total angle traversed and solid angle covered. In what follows, we introduce the WAIT AND CHECK ALGORITHM (cf. Table 4.5). This algorithm has a much better performance with regards to solid angle covered, at the expense of a longer execution time. After establishing its correctness in Theorem 4.1.17, we show how to modify it to achieve an optimal total rotation given its solid angle covered (cf. Remark 4.1.18).

The next lemma will be used in establishing the correctness of the WAIT AND CHECK ALGORITHM.

Name:	WAIT AND CHECK ALGORITHM
Goal:	Solve the formation initialization problem using near-optimal solid angle coverage.
Assumes:	(i) Spacecraft model in Section 4.1.1. (ii) $\Theta_{\text{fov}} > \frac{\pi}{4}$.

1: Define $\Theta_\epsilon = \Theta_{\text{fov}} - \frac{\pi}{4}$
2: **if** $S_i \in G_1$ **then**
3: Rotate to align M_{S_i} with I_3
4: **else**
5: Rotate to align M_{S_i} with M_{opp}
6: **end if**
7: Wait for common start time t_s
8: Rotate by $\frac{\pi}{4}$ about Y_{S_i} {Call the time at the end of this step t_1}
9: Rotate about X_{S_i} by 2π with angular velocity ω {Call this time t_2}
10: Wait $\frac{\tan(\frac{\pi}{2}-\Theta_\epsilon)}{\Theta_\epsilon}(t_2 - t_1)$ {Call this time t_3}
11: Rotate about Y_{S_i} by $\frac{-\pi}{2}$ {Call this time t_4}
12: Rotate about X_{S_i} by 3π with angular velocity ω {Call this time t_5}
13: Rotate about Y_{S_i} by $\frac{-\pi}{2}$ {Call this time t_6}
14: Wait $\frac{\tan(\frac{\pi}{2}-\Theta_\epsilon)}{\Theta_\epsilon}(t_5 - t_1)$ {Call this time t_1}
15: Rotate about X_{S_i} by 2π with angular velocity ω {Call this time t_7}

Table 4.5: The WAIT AND CHECK ALGORITHM.

Lemma 4.1.16. *Consider a spacecraft S_2 traveling in a path with respect to S_1 with velocity V_{S_2} and point of closest approach $p_{cl}(S_1, S_2)$. Let $\Pi_{1,2}$ be the plane in $CMF(S_1)$ spanned by the vectors $p_{cl}(S_1, S_2)$ and V_{S_2}. Define a parametrization of vectors in $\Pi_{1,2}$ by the function $\Theta_{scan}(P) = \arctan(p_{cl}(S_1, S_2) \cdot P, -V_{S_2} \cdot P)$. For any angles $\Theta \in [0, \pi]$ and $\epsilon \in [0, \Theta]$, if S_1 first verifies that $\Theta_{scan}(P_{S_2}) < \Theta - \epsilon$ at time t_1 and then verifies that $\Theta_{scan}(P_{S_2}) > \Theta + \epsilon$ at time t_2, then by time $t_2 + \frac{\tan(\frac{\pi}{2}-\epsilon)}{\epsilon}(t_2 - t_1)$, S_2 will be within ϵ of its final angle.*

Proof. Since $\Theta_{scan}(P_{S_2}(t2)) - \Theta_{scan}(P_{S_2}(t1)) > 2\epsilon$, $\frac{\|V_{S_2}\|}{\|p_{\text{cl}}(S_1,S_2)\|}$ is at least $\frac{2\epsilon}{t_2-t_1}$. $\Theta_{scan}(P_{S_2}(\frac{\tan(\frac{\pi}{2}-\epsilon)}{\epsilon}(t_2 - t_1))) \geq \arctan(\tan(\Theta + \epsilon) + \frac{2\epsilon}{t_2-t_1}\frac{\tan(\frac{\pi}{2}-\epsilon)}{\epsilon}(t_2 - t_1)) \geq \pi - \epsilon$. □

Next, we characterize the correctness of the WAIT AND CHECK ALGORITHM.

Theorem 4.1.17. *The* WAIT AND CHECK ALGORITHM *correctly solves the formation initialization problem.*

Proof. Consider a spacecraft, S_1. Any other spacecraft whose X position is less than zero at time t_1 must either be found, cross the $\{Y, Z\}$ plane, or cross the $\{X, Z\}$ plane before t_2. If S_2 crossed the $\{X, Z\}$ plane between t_1 and t_2 and was not found, then it must have been moving with sufficient velocity to have moved to within Θ_ϵ of its final angle by time t_3. By this logic, by t_3, any craft with a final angle corresponding to a positive X component of position must have been found by time t_2, or be on the $+X$ side of the $\{Y, Z\}$ plane by time t_3. Between t_4 and t_5 all such craft are found, along with any craft that started on the $+X$ side of the $\{Y, Z\}$ plane and have not left it by t_5 (by Lemma 4.1.16). Any craft which have left the $+X$ side of the $\{Y, Z\}$ plane by t_5 but were not found during the sweep of the $-X$ half of the $\{Y, Z\}$ plane must have been moving with sufficient angular velocity as to be within Θ_ϵ of their final angles (on the $-X$ half of the $\{Y, Z\}$ plane) by t_6 (cf. Lemma 4.1.16). For this reason, the final sweep of the $-X$ side of the $\{Y, Z\}$ plane need only be a 2π sweep. □

Remark 4.1.18 (Angle-optimal region sweeps)**.** *The* WAIT AND CHECK ALGORITHM *covers a solid angle of* $7\pi + \frac{5\Theta_{tilt}}{\sin(\Theta_{tilt})}$*. Clearly, the ratio of total angle traversed to solid angle covered in the* WAIT AND CHECK ALGORITHM *is not at the optimal* $\frac{1}{2\sin(\Theta_{fov})}$*. The algorithm can be modified to traverse a total angle of* $7\pi\sin(\Theta_{tilt}) + 5\Theta_{tilt}$*, where* $\Theta_{tilt} = \min(\pi/2 - \Theta_{fov}, \Theta_{sun}, \Theta_{fov})$*, at the expense of not respecting the sun-angle constraint. We describe how next. The optimal ratio of total angle traversed to solid angle covered is achievable for any rotational trajectory of* $\vec{v}_{SENSOR}(S)$ *over time. While a rotational velocity,* ω*, specifies the instantaneous rotation of the entire body frame of* S*, the instantaneous motion of* $\vec{v}_{SENSOR}(S)$ *only fixes two degrees of freedom of this rotation. By choosing* ω *to lie along* $\vec{v}_{SENSOR}(S) \times \frac{d}{dt}\vec{v}_{SENSOR}(S)$*, we can always achieve the maximum instantaneous* $f_{sld}(\omega)/\|\omega\|$*.*

Let us suppose that $\vec{v}_{SENSOR}(S)$ *is within an angle of* $\frac{\pi}{2} - \alpha$ *of the sun line, and we wish for* $\vec{v}_{SENSOR}(S)$ *to sweep out the arc defined by* $C_\alpha = \{\vec{v} \in \mathbb{R}^3 \ : \ \|\vec{v}\| = 1 \wedge \arccos(\vec{v} \cdot \vec{v}_{SUN}) = \frac{\pi}{2} - \alpha\}$*. At any instant during which* $\vec{v}_{SENSOR}(S) \in C_\alpha$*, the optimal axis of rotation,* ω*, is both perpendicular to* $\vec{v}_{SENSOR}(S)$ *and guarantees* $\vec{v}_{SENSOR}(S)$ *remains in* C_α*. One such* ω *always lies on a cone which we will define as* $C_{tumble} = \{\vec{v} \in \mathbb{R}^3 \ : \ \|\vec{v}\| = 1 \wedge \arccos(\vec{v} \cdot \vec{v}_{SUN}) = \alpha\}$*, see Figure 4.4. Note that the body frame,* $BF(S)$ *does not move with respect to* $CMF(S)$ *at any point along the axis* ω*. When the sweeps about the sun line of the* WAIT AND CHECK ALGORITHM *are executed as we just described, the algorithm requires a total angular rotation of* $\frac{7\pi}{\sqrt{2}} + 5\Theta_{tilt}$*.* •

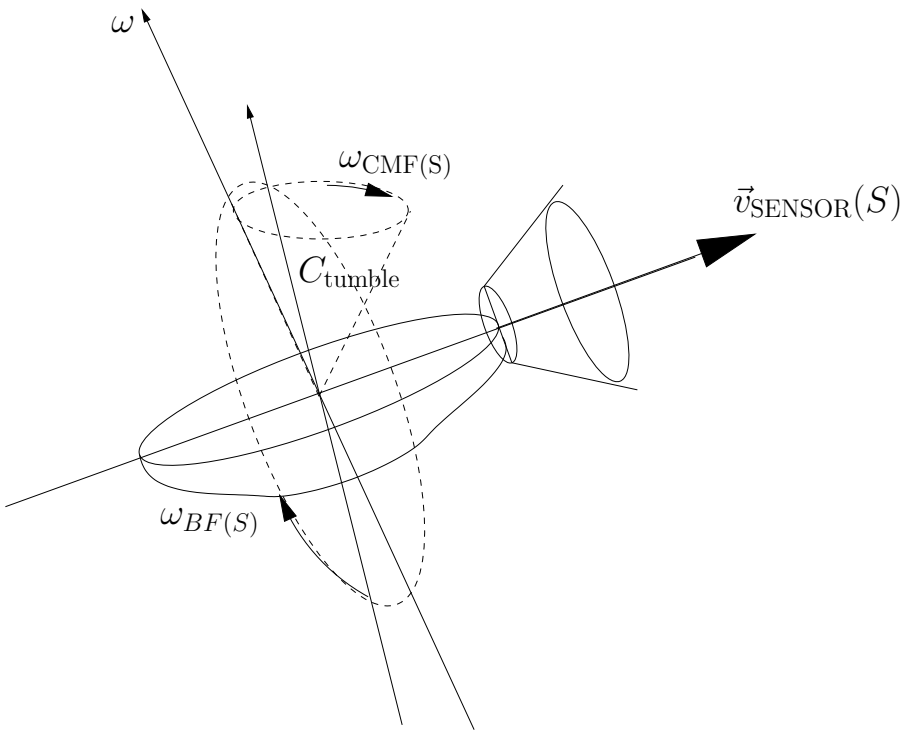


Figure 4.4: Performing a sweep of 2π with less then 2π rotation

Part III

Instantaneous connectivity of graphs induced by inter-agent distance

Chapter 5

Connectivity maintenance solution

5.1 The Connectivity Maintenance Algorithm

This section introduces the Connectivity Maintenance (abbreviated CM) Algorithm. By itself, the algorithm does not invoke either physical agents or their mobility, and fits within common frameworks of distributed algorithms.

5.1.1 Algorithm description

Given a coordination task and a motion coordination algorithm to achieve it, maintaining a fixed set spanning tree throughout the evolution will guarantee connectivity preservation but, in general, will interfere in the optimal achievement of the task. The Connectivity Maintenance Algorithm is a procedure that, coupled with individual motion control strategies of the network agents, maintains an evolving spanning tree of the communication graph. The underlying idea is that if the motion of the robots is constrained to not break any links of the spanning tree, the communication graph will remain connected as well. Let us begin by describing the algorithm informally.

[Informal description:] Each robot maintains a reference to its parent in the spanning tree.

At pre-arranged times, each robot is allowed to change its parent, in accordance with a set of preferences specified in suitable way (normally, by a motion coordination algorithm). Connectivity is then preserved as follows. Each robot keeps an estimate of its depth, i.e., the distance to the root in the spanning tree. If no robot picks a robot of greater depth than its parent's, then no robot will pick one of its current descendants as a parent node. To allow robots to attach to potential parents of the same depth estimate, a tie-breaking algorithm based on UIDs is used to prevent the formation of cycles, thus maintaining the spanning tree property.

5.1.1.1 Formal definition

Next, we provide a formal definition using the formalism described in Section 2.1. Given a network $\mathcal{S}$ with communication links determined by proximity graph $\mathcal{G}$, the Connectivity Maintenance Algorithm is an input-output control and communication law $\mathcal{CC}$ for $\mathcal{S}$ consisting of the sets:

(i) $L = W$;

(ii) $W = \mathbb{N} \times \mathbb{Z}_4 \times \mathbb{Z}_n^3 \times \mathbb{Z}_n \cup \{\emptyset\} \times \mathbb{Z}_2$, are sets of values of the variables

$w^{[i]} = (\text{dpth}_{\text{est}}^{[i]}, \text{phase}^{[i]}, f_{\text{p}}^{[i]}, g_{\text{p}}^{[i]}, n_{\text{root}}^{[i]}, n_{\text{dep-targ}}^{[i]}, \mathbb{I}_{\text{par-less}}^{[i]})$,

for $i \in \mathbb{Z}_n$.

The meaning of the components of $w^{[i]}$ is as follows. $\text{dpth}_{\text{est}}^{[i]}$ is a depth estimate. $\text{phase}^{[i]}$ is a round counter indicating the current mode of the algorithm. $f_{\text{p}}^{[i]}$ is a parent identifier. $g_{\text{p}}^{[i]}$ is a proposed next parent. $n_{\text{root}}^{[i]}$ indicates the UID of the root of the tree containing agent i. $n_{\text{dep-targ}}^{[i]}$ is the UID of an agent which i would like to attach to as a parent. It is the most preferred parent among those which the algorithms coupled with CM Algorithm allow (more details are provided below when we discuss $W_{\text{in}}{}^{[i]}$). Note that i may not be allowed by the algorithm to propose that $n_{\text{dep-targ}}^{[i]}$ be its next parent, for which reason $g_{\text{p}}^{[i]}$ might not equal $n_{\text{dep-targ}}^{[i]}$. Finally, $\mathbb{I}_{\text{par-less}}^{[i]}$ is a Boolean indicator denoting whether i's parent $f_{\text{p}}^{[i]}$ had a strictly smaller depth estimate than i as of the most recent communication round.

If $n^{[i]}_{\text{root}}$ is different from 0 for some agent i, then the tree needs to be repaired. $n^{[i]}_{\text{dep-targ}}$ is also used as an output logic variable, signaling that robot i wishes to increase $\text{dpth}^{[i]}_{\text{est}}$ with the goal of attaching to $n^{[i]}_{\text{dep-targ}}$;

(iii) $W^{[i]}_0 = \{(0, 0, i, i, i, \emptyset, \text{false})\} \subseteq W$, $i \in \mathbb{Z}_n$. Note that $f^{[i]}_{\text{p}} = i$ and $\text{dpth}^{[i]}_{\text{est}} = 0$ for $i \in \mathbb{Z}_n$. Also note that the tree is initially in need of repair;

(iv) W_{in}, are sets of values of input variables, ${w_{\text{in}}}^{[i]}$, $i \in \mathbb{Z}_n$, given by

$$\begin{aligned} {w_{\text{in}}}^{[i]} =& (\mathbb{I}^{[i]}_{\text{incr-dep}}, n^{[i]}_{\text{incr-sgnl}}, R^{[i]}_{\text{pref}}, f^{[i]}_{\text{allow}}) \in \\ & \mathbb{Z}_2 \times \mathbb{Z}_n \cup \{\emptyset\} \times R(\mathbb{Z}_n, \mathbb{Z}_n) \times \mathfrak{F}(\mathbb{Z}_n; \{\text{true}, \text{false}\}). \end{aligned}$$

The meaning of the components of w_{in} is as follows. $\mathbb{I}^{[i]}_{\text{incr-dep}}$ is a Boolean variable that determines whether the agent's depth estimate update is in a special mode to allow for more flexible re-arrangements. $n^{[i]}_{\text{incr-sgnl}}$ is the identity of an agent which is in the special mode indicated by $\mathbb{I}^{[\cdot]}_{\text{incr-dep}}$. $R^{[i]}_{\text{pref}}$ is a strict order relation ranking the order in which node i would prefer to attach to each other node as its parent: $(j, k) \in R^{[i]}_{\text{pref}}$ means node i would prefer to attach to j over k. We stipulate that each $R^{[i]}_{\text{pref}}$ must satisfy $(j, i) \in R^{[i]}_{\text{pref}}$ for all $j \in \{0, \ldots, n-1\} \setminus \{i\}$ and that $n^{[j]}_{\text{root}} < n^{[k]}_{\text{root}}$ implies $(j, k) \in R^{[i]}_{\text{pref}}$. $R^{[i]}_{\text{pref}}$ can be thought of as a symbol string representing computation to be performed to evaluate whether $(j, k) \in R^{[i]}_{\text{pref}}$. Finally, $f^{[i]}_{\text{allow}} : \mathbb{Z}_n \to \{\text{true}, \text{false}\}$ is a function which maps a node j to true if and only if the algorithm supplying $f^{[i]}_{\text{allow}}$ will allow $\text{dpth}^{[i]}_{\text{est}}$ to increase in order for i to attach to j under the CM Algorithm;

(v) ${W_{\text{in},0}}^{[i]} = \{(\text{false}, \emptyset, R_{i-\text{init}}, f^{[i]}_{\text{allow}})\}$, where $(j, k) \in R_{i-\text{init}}$ if $n^{[j]}_{\text{root}} < n^{[k]}_{\text{root}}$ or $n^{[j]}_{\text{root}} = n^{[k]}_{\text{root}}$ and $((k = i \text{ and } j \neq i) \text{ or } j < k)$, and $f^{[i]}_{\text{allow}} \equiv \text{false}$;

(vi) $W_{\text{out}} = \mathbb{Z}_n \times \mathbb{F}(\mathbb{Z}_n) \times \mathbb{F}(\mathbb{Z}_n) \times \mathbb{Z}_n \cup \{\emptyset\}$ are sets of values of output variables

$w_{\text{out}}{}^{[i]} = (f_{\text{p}}^{[i]}, S_{\text{constraints}}^{[i]}, S_{\text{children}}^{[i]}, n_{\text{dep-targ}}^{[i]})$,

for $i \in \mathbb{Z}_n$.

The meaning of the components of W_{out} is as follows. $f_{\text{p}}^{[i]}$ (taken from $W^{[i]}$) specifies the parent of agent i to the motion control algorithm. $S_{\text{constraints}}^{[i]} = \{j \in \mathbb{Z}_n \ : \ n_{\text{root}}^{[j]} \neq n_{\text{root}}^{[i]}\}$ is the set of agents whose estimate of their root nodes are different from those of i and $S_{\text{children}}^{[i]} = \{j \in \mathbb{Z}_n \ : \ f_{\text{p}}^{[j]} = i\}$ are the children of i in the constraint tree. If every i maintains connectivity with each agent in $\{f_{\text{p}}^{[i]}\} \cup S_{\text{constraints}}^{[i]} \cup S_{\text{children}}^{[i]}$, then the communication graph remains connected. $n_{\text{dep-targ}}^{[i]} \in \mathbb{Z}_n \cup \{\emptyset\}$ is copied from W and used to signal which agents i wants to increase depth to connect to;

and of the maps:

(i) **function** $\text{msg}(x, w, w_{\text{in}}, j) = w$;

(ii) **function** $\text{stf}_{\text{io-slf}}(w, w_{\text{in}}, l) =$

1: **if** $n_{\text{root}}^{[\text{id}]} = \text{id}$ **then**

2: Set $n_{\max} \leftarrow \text{id}(n+1)$

3: **else**

4: Set $n_{\max} \leftarrow n_{\text{root}}^{[\text{id}]}(n+1) + n$

5: **end if**{*If proposed parent has same depth as id, is trying to change its parent, and idis greater than my proposed parent's UID and the UIDs of any nodes that have proposed attaching to me as a parent, then keep current parent. Otherwise choose proposed parent*}

6: **if** $\text{phase}^{[\text{id}]} = 0$ **then**

7: **if** $g_{\text{p}}^{[g_{\text{p}}^{[\text{id}]}]} \neq f_{\text{p}}^{[g_{\text{p}}^{[\text{id}]}]}$, $\text{dpth}_{\text{est}}^{[g_{\text{p}}^{[\text{id}]}]} = \text{dpth}_{\text{est}}^{[\text{id}]}$ and $\text{id} > \max\{j \ : \ g_{\text{p}}^{[j]} = \text{id} \vee j = g_{\text{p}}^{[\text{id}]}\}$ **then**

Set $g_{\mathrm{p}}^{[\mathrm{id}]} \leftarrow f_{\mathrm{p}}^{[\mathrm{id}]}$

else

Set $f_{\mathrm{p}}^{[\mathrm{id}]} \leftarrow g_{\mathrm{p}}^{[\mathrm{id}]}$ {*We call this the "re-attach" step*}

end if

end if

if $\mathrm{phase}^{[\mathrm{id}]} = 1$ {*Update depth estimate*} **then**

if $\mathbb{I}_{\text{incr-dep}}^{[i]} = \text{false}$ **then**

Set $\mathrm{dpth}_{\mathrm{est}}^{[\mathrm{id}]} \leftarrow \min(\mathrm{dpth}_{\mathrm{est}}^{[f_{\mathrm{p}}^{[\mathrm{id}]}]} + 1, n_{\max})$

else

Set $\mathrm{dpth}_{\mathrm{est}}^{[\mathrm{id}]} \leftarrow \min(\mathrm{dpth}_{\mathrm{est}}^{[\mathrm{id}]} + 1, n_{\max})$

end if

end if

if $\mathrm{phase}^{[\mathrm{id}]} = 2$ **then**

Set $\mathbb{I}_{\text{par-less}}^{[\mathrm{id}]} \leftarrow \text{false}$

if $\mathrm{dpth}_{\mathrm{est}}^{[f_{\mathrm{p}}^{[\mathrm{id}]}]} < \mathrm{dpth}_{\mathrm{est}}^{[\mathrm{id}]}$ **then**

Set $\mathbb{I}_{\text{par-less}}^{[\mathrm{id}]} \leftarrow \text{true}$

end if

end if{*As a worst-case solution, consider attaching to yourself. By definition of* $R_{pref}^{[i]}$*, this does not happen unless there are no other options*}

if $\mathrm{phase}^{[\mathrm{id}]} = 3$ {*Here we pick* $g_p^{[i]}$ *and* $n_{dep\text{-}targ}^{[i]}$ *according to* $R_{pref}^{[i]}$ *among the appropriate sets of allowable values*} **then**

Let $S_< = \{j \in \mathcal{N}(\mathrm{id}) \ : \ \mathrm{dpth}_{\mathrm{est}}^{[j]} < \mathrm{dpth}_{\mathrm{est}}^{[\mathrm{id}]}$ or $f_{\mathrm{p}}^{[j]} = f_{\mathrm{p}}^{[\mathrm{id}]}$ or $\mathrm{dpth}_{\mathrm{est}}^{[j]} = \mathrm{dpth}_{\mathrm{est}}^{[\mathrm{id}]}$ and $\mathbb{I}_{\text{par-less}}^{[j]} =$ true or $j = i\}$

28: Let $g_{\mathrm{p}}^{[\mathrm{id}]} \leftarrow$ some element of $\{j \in S_{<} \ : \nexists k, (k,j) \in R_{\mathrm{pref}}^{[\mathrm{id}]}, k \in S_{<}\}$

29: **if** $\{j \in S_{<} \ : \nexists k, (k,j) \in R_{\mathrm{pref}}^{[\mathrm{id}]}, k \in S_{<}\} = \{f_{\mathrm{p}}^{[\mathrm{id}]}\}$ **then**

30: Let $S_{\mathrm{sgnl}} \leftarrow \{j \in \mathcal{N}(\mathrm{id}) \ : \ f_{\mathrm{allow}}^{[i]}(j) = \mathrm{true}\}$

31: Let $n_{\text{dep-targ}}^{[\mathrm{id}]} \leftarrow \{j \in S_{\mathrm{sgnl}} \ : \nexists k \in S_{\mathrm{sgnl}}, (k,j) \in R_{\mathrm{pref}}^{[\mathrm{id}]}\}$

32: **if** $n_{\text{dep-targ}}^{[\mathrm{id}]} = \mathrm{id}$ or $n_{\text{dep-targ}}^{[\mathrm{id}]} = f_{\mathrm{p}}^{[\mathrm{id}]}$ **then**

33: $n_{\text{dep-targ}}^{[\mathrm{id}]} \leftarrow$ some element of $\{j \in \mathcal{N}(\mathrm{id}) \ : \nexists k \in \mathcal{N}(\mathrm{id}), (k,j) \in R_{\mathrm{pref}}^{[\mathrm{id}]}\}$

34: **end if**

35: **else**

36: $n_{\text{dep-targ}}^{[\mathrm{id}]} \leftarrow \emptyset$

37: **end if**

38: **end if**

39: **if** $f_{\mathrm{p}}^{[\mathrm{id}]} = \mathrm{id}$ **then**

40: Set $n_{\mathrm{root}}^{[\mathrm{id}]} \leftarrow \mathrm{id}$

41: **else**

42: Set $n_{\mathrm{root}}^{[\mathrm{id}]} \leftarrow n_{\mathrm{root}}^{[f_{\mathrm{p}}^{[\mathrm{id}]}]}$

43: **end if**

44: Set $\mathrm{phase}^{[\mathrm{id}]} \leftarrow (\mathrm{phase}^{[\mathrm{id}]} + 1) \bmod 4$

45: return $(\mathrm{dpth}_{\mathrm{est}}^{[\mathrm{id}]}, \mathrm{phase}^{[\mathrm{id}]}, f_{\mathrm{p}}^{[\mathrm{id}]}, g_{\mathrm{p}}^{[\mathrm{id}]}, n_{\mathrm{root}}^{[\mathrm{id}]}, n_{\text{dep-targ}}^{[\mathrm{id}]}, \mathbb{I}_{\text{par-less}}^{[\mathrm{id}]})$

In the update step of $\mathrm{phase}^{[\mathrm{id}]} = 2$, there are two possible depth update steps, one on line `12:` and the other on line `14:`. Line `12:` is an update rule which, if universally followed, causes each agent's depth estimate, $\mathrm{dpth}_{\mathrm{est}}^{[\cdot]}$, to converge on its actual depth in the tree. Line `14:` is used when one agent wants to deliberately increase its depth estimate above its actual depth in order to attach to another desired agent. Usage of line`14:` will be discussed in more detail in Section 5.4.1. Note

that `9:` is labelled as the "re-attach" step. We will use this nomenclature throughout the paper;

(iii) **function** $\mathrm{stf}_{\text{io-out}}(w, w_{\text{in}}, l) = (f_{\text{p}}^{[\text{id}]}, \{j \in \mathcal{N}(i) \ : \ n_{\text{root}}^{[j]} \neq n_{\text{root}}^{[i]}\}, \{j \in \mathcal{N}(i) \ : \ f_{\text{p}}^{[j]} = i\}, n_{\text{dep-targ}}^{[i]})$.

(iv) $\mathrm{ctl}(x_{t_\ell}, x, w, w_{\text{in}}) = 0$.

5.1.2 Depth-compatible and motion-compatible algorithms

We intend for the Connectivity Maintenance Algorithm to be coupled with two classes of input-output laws. The first class of algorithms, that we term depth-compatible, specify the update depth rule to be used by the Connectivity Maintenance Algorithm by setting the value of the variable $\mathbb{I}_{\text{incr-dep}}^{[i]}$ for each agent after receiving the value of $n_{\text{dep-targ}}^{[i]}$. The second class of algorithms, that we term motion-compatible, specify the dynamics of the network agents subject to connectivity constraints and specifies the set of preferences $R_{\text{pref}}^{[i]}$ for each agent.

We begin by formalizing the notion of depth-compatible algorithm.

Definition 5.1.1. *An input-output control and communication law* CC_d *is* depth compatible with CM *if the following hold:*

- $W_{in} = \mathbb{Z}_n \cup \{\emptyset\}$ *are sets of values of* input logic variables $w_{in}{}^{[i]} = n_{dep\text{-}targ}^{[i]}$ *for* $i \in I$*;*
- W_{out}*, are sets of values of* output logic variables, $(\mathbb{I}_{incr\text{-}dep}^{[i]}, n_{incr\text{-}sgnl}^{[i]}, f_{allow}^{[i]}) \in \mathbb{Z}_2 \times \mathbb{Z}_n \cup \{\emptyset\} \times \mathfrak{F}(\mathbb{Z}_N; \{true, false\})$*;*
- *The composition with* CM Algorithm *guarantees that the maximum depth estimate* $dpth_{est}^{[\cdot]}$ *that any agent* i *having* $n_{root}^{[i]} = 0$ *holds for more than 4 rounds is bounded from above by a function of the number of agents* n*.*

Combining an algorithm which is *depth compatible with* CM yields a new input-output control and communication law. The simplest algorithm which is *depth compatible with* CM, and the one

we assume unless otherwise stated, is NULL DEPTH INCREMENT ALGORITHM. Essentially, the only variables that NULL DEPTH INCREMENT ALGORITHM specifies are $\mathbb{I}^{[i]}_{\text{incr-dep}} = \text{false}$ and $f^{[i]}_{\text{allow}} \equiv \text{false}$ for all $i \in \mathbb{Z}_n$. A formal definition of NULL DEPTH INCREMENT ALGORITHM can be given in the form of an input-output control and communication law as in Definition 2.2.1, but we omit it for brevity.

Note that we cap the depth estimate of any given node at a number (denoted by the temporary variable $n_{\max}$ within the description of CM) determined by the number of agents. We show that this does not affect the operation under NULL DEPTH INCREMENT ALGORITHM in the next result.

Theorem 5.1.2. *If $\mathbb{I}^{[i]}_{incr\text{-}dep} = false$ and $n^{[i]}_{root} = 0$ for all $i \in \mathbb{Z}_n$, the following invariant is maintained: at most $n - k$ nodes have depth estimates greater than or equal to k at any time.*

Proof. Assume the result holds for round t. To show the invariant holds at round $t+1$, we induct on the depth estimate, using as the base case the fact that at most n nodes have depth estimate 0. Let any k nodes, $i_i, \ldots, i_k$ each increase depth (i.e., $\text{dpth}^{[i_j]}_{\text{est}}(t+1) > \text{dpth}^{[i_j]}_{\text{est}}(t)$ for all $j \in \{1, \ldots, k\}$) according to $\text{dpth}^{[i_j]}_{\text{est}}(t+1) \leftarrow \text{dpth}^{[f^{[i_j]}_{\text{p}}]}_{\text{est}}(t) + 1$, $j \in \{1, \ldots, k\}$ ($n_{\max}$ does not appear, as it is greater than the maximum value of $\text{dpth}^{[\cdot]}_{\text{est}}$ of any agent we are considering). To have greater depth at $t+1$ than on round t, each i_j must have had a parent, $f^{[i_j]}_{\text{p}}$, having $\text{dpth}^{[f^{[i_j]}_{\text{p}}]}_{\text{est}}(t) = \text{dpth}^{[i_j]}_{\text{est}}(t)$. Without loss of generality, let i_1 be the agent with the least depth at round t. By our induction hypothesis, there were at most $n - \text{dpth}^{[i_1]}_{\text{est}}(t)$ nodes at depth $\text{dpth}^{[i_1]}_{\text{est}}(t)$ or greater on round t. Thus, there were at most $n - \text{dpth}^{[i_1]}_{\text{est}}(t) - k - 1$ already at depth estimates $\geq \text{dpth}^{[i_1]}_{\text{est}}(t)$, since k must have been at depth estimate $\text{dpth}^{[i_1]}_{\text{est}}$ to increase at round $t+1$, and at least one node must have been at depth $\text{dpth}^{[f^{[i_1]}_{\text{p}}]}_{\text{est}}$. This gives us at most $n - \text{dpth}^{[i_1]}_{\text{est}}(t) - k - 1 + k$ nodes at depth $\text{dpth}^{[i_1]}_{\text{est}}(t) + 1$ or greater at time $t+1$. □

The operation of CM ALGORITHM composed with NULL DEPTH INCREMENT ALGORITHM is illustrated in Figure 5.1.

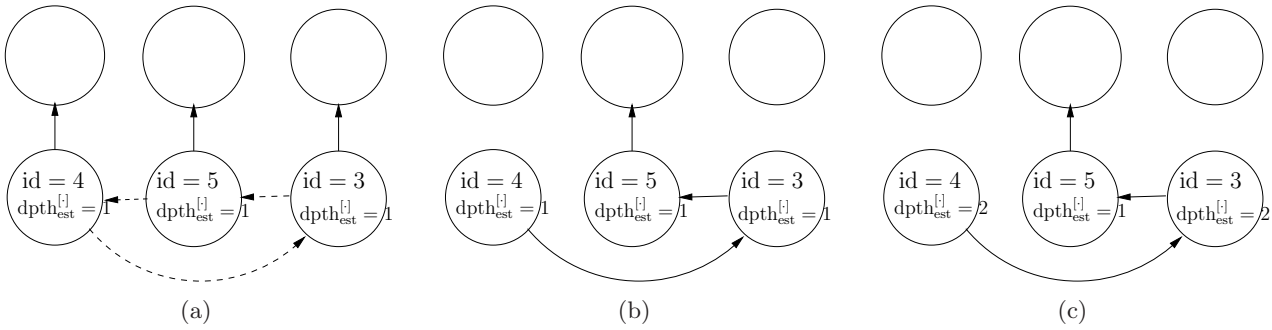


Figure 5.1: Illustration of CM Algorithm combined with Null Depth Increment Algorithm. Frame (a) indicates connections of the form $(i, f_{\mathrm{p}}^{[i]})$ as solid arrows and $(i, g_{\mathrm{p}}^{[i]})$ as dashed arrows. Frame (b) illustrates the constraint tree after steps `5:` through `9:` of CM Algorithm. Frame (c) shows the result of the depth update in lines `10:` through `14:`.

We end this section by formalizing the notion of motion-compatible algorithms. As we will see later, certain properties of the dynamics of the communication graph need to hold for the correctness properties of CM Algorithm to hold, and these are captured in the following notion.

Definition 5.1.3. *An input-output control and communication law* CC_d *is* motion compatible with CM *if the following hold:*

- $W_{in} = \mathbb{Z}_n \times \mathbb{F}(\mathbb{Z}_n) \times \mathbb{F}(\mathbb{Z}_n) \times \mathbb{Z}_n \cup \{\emptyset\}$, $w_{in}{}^{[i]} = (f_p^{[i]}, S^{[i]}_{constraints}, S^{[i]}_{children}, n^{[i]}_{dep\text{-}targ})$ *for* $i \in I$*;*
- $W_{out} = R(\mathbb{Z}_n, \mathbb{Z}_n)$, $w_{out}{}^{[i]} = R^{[i]}_{pref}$, *for* $i \in I$*;*
- *When combined with* CM Algorithm*, the following hold: the time schedule, output state transition function, and control function of* CC_d *are such that the control function is guaranteed never to induce a motion which causes* $(i, g_p^{[i]})$ *or* $(i, f_p^{[i]})$ *to cease to be an edge of the underlying proximity graph. We require that these also guarantee that no edge,* (i, j)*, of the underlying graph, having* $n^{[i]}_{root} \neq n^{[j]}_{root}$*, ever be broken. Note that* $S^{[i]}_{constraints}$ *is exactly* $\{j \in \mathcal{N}(i) \,:\, n^{[j]}_{root} \neq n^{[i]}_{root}\}$*.*

5.2 Correctness analysis

In this section, we analyze the correctness of Connectivity Maintenance Algorithm when combined with any algorithm which is *motion compatible with* CM and any algorithm which is *depth compatible with* CM. We show that connectivity is preserved throughout the execution of the algorithm and also analyze its tree repair properties. We start by characterizing the topological properties of the constraint tree under Connectivity Maintenance Algorithm.

For convenience, in the forthcoming analysis, we let $\mathrm{rnd}(t) \in \mathbb{N}$ be the number of times the assignment $\mathrm{phase}^{[i]} \leftarrow 2$ has been made at time t. We denote the value of, say $\mathrm{dpth}^{[i]}_{\mathrm{est}}$ at iteration $\mathrm{rnd}(t)$, by $\mathrm{dpth}^{[i]}_{\mathrm{est}}(\mathrm{rnd}(t))$.

Theorem 5.2.1. *The execution of the* CM Algorithm *verifies that*

(i) $dpth^{[i]}_{est}(rnd(t)) \leq dpth^{[i]}_{est}(rnd(t)-1)+1$,

(ii) $dpth^{[i]}_{est}(rnd(t)) \geq dpth^{[f^{[i]}_p]}_{est}(rnd(t))$,

for $i \in \mathbb{Z}_n$, *where for convenience,* $dpth^{[i]}_{est}(r) = dpth^{[i]}_{T}(t_0)$ *for all rounds* $r \leq 0$. *Thus, at any time* $t \geq 0$, *if* k *is an ancestor of* i, *then* $dpth^{[i]}_{est}(rnd(t)) \geq dpth^{[k]}_{est}(rnd(t))$.

Proof. Note that either lines `12:` or `14:` of $\mathrm{stf}_{\mathrm{io-slf}}$ for CM Algorithm, cf. Section 5.1.1.1, are the only steps where the value of $\mathrm{dpth}^{[i]}_{\mathrm{est}}$ is modified. We refer to either of these steps as the *update rule*. Step `14:` trivially maintains *(i)*. To show `12:` also maintains *(i)*, we induct on the current round, $\mathrm{rnd}(t)$. Let j be $f^{[i]}_{\mathrm{p}}$ at iteration $\mathrm{rnd}(t)$. This can only happen because either (a) j became i's parent due to a re-attach or (b) j was i's parent at $\mathrm{rnd}(t)-1$. In case (a), the re-attach requires $\mathrm{dpth}^{[j]}_{\mathrm{est}}(\mathrm{rnd}(t)-1) \leq \mathrm{dpth}^{[i]}_{\mathrm{est}}(\mathrm{rnd}(t)-1)$, and this implies that $\mathrm{dpth}^{[i]}_{\mathrm{est}}(\mathrm{rnd}(t)) \leq \mathrm{dpth}^{[i]}_{\mathrm{est}}(\mathrm{rnd}(t)-1)+1$. In case (b), $\mathrm{dpth}^{[i]}_{\mathrm{est}}(\mathrm{rnd}(t)-1) = \mathrm{dpth}^{[j]}_{\mathrm{est}}(\mathrm{rnd}(t)-2)+1$. The induction hypothesis implies that

$\text{dpth}^{[j]}_{\text{est}}(\text{rnd}(t)-1) \leq \text{dpth}^{[j]}_{\text{est}}(\text{rnd}(t)-2)+1$, and therefore $\text{dpth}^{[i]}_{\text{est}}(\text{rnd}(t)) \leq \text{dpth}^{[i]}_{\text{est}}(\text{rnd}(t)-1)+1$. To prove *(ii)*, we note that either i has attached to $f_{\text{p}}^{[i]}$ more recently than $\text{dpth}^{[f_{\text{p}}^{[i]}]}_{\text{est}}$ has changed, or, by the update rule, $\text{dpth}^{[i]}_{\text{est}}(\text{rnd}(t)) \geq \text{dpth}^{[f_{\text{p}}^{[i]}]}_{\text{est}}(\text{rnd}(t)-1)+1$, from which we get $\text{dpth}^{[i]}_{\text{est}}(\text{rnd}(t)) \geq \text{dpth}^{[f_{\text{p}}^{[i]}]}_{\text{est}}(\text{rnd}(t)$, which concludes the result. □

The above result is key in showing that if the constraint tree begins with k connected components, then, under certain technical conditions, the CM Algorithm is guaranteed to preserve their connectivity. In particular, if the constraint tree begins connected, $k=1$, then it will remain connected throughout the execution of the algorithm,

Theorem 5.2.2. *Assume the graph induced by $(i, f_p^{[i]}), i \in \mathbb{Z}_n$ starts with k disjoint connected components. Then, at all times during the execution of* CM Algorithm*, the graph induced by the parent relation among the agents contains no cycles other than those it started with (and, having n edges, retains at most k disjoint connected components), so long as no edge of the form $(i, f_p^{[i]})$ or $(i, g_p^{[i]})$ disappears from the underlying graph.*

Proof. Obviously the removal of an edge of the form $(i, f_{\text{p}}^{[i]})$ causes the algorithm to set $f_{\text{p}}^{[i]} \leftarrow i$, potentially introducing a new cycle. Barring this possibility, let us proceed with the rest of the proof. We assume the introduction of a new cycle and proceed by contradiction. Any cycle in the graph must consist entirely of agents having the same depth estimate (this follows from $\text{dpth}^{[i]}_{\text{est}} \geq \text{dpth}^{[f_{\text{p}}^{[i]}]}_{\text{est}}$ from Theorem 5.2.1 and the fact that for a cycle C, we have $\sum_{i \in C} \text{dpth}^{[i]}_{\text{est}} - \text{dpth}^{[f_{\text{p}}^{[i]}]}_{\text{est}} = 0$). During the four communication rounds, cf. Section 5.1.1.1, leading up to the creation of any cycle, all agents involved in the cycle must have the same depth estimate as well, otherwise there exists either an agent $i \in C$ having $\text{dpth}^{[f_{\text{p}}^{[i]}]}_{\text{est}} > \text{dpth}^{[i]}_{\text{est}}$, which violates Theorem 5.2.1, or an agent $i \in C$ having $\text{dpth}^{[g_{\text{p}}^{[i]}]}_{\text{est}} > \text{dpth}^{[i]}_{\text{est}}$, a choice which is prohibited by step `20:` of the CM Algorithm. When a cycle first appears, it must appear due to some agent i connecting to a new parent j with the same depth estimate. However, this

new parent must, in turn, attach to an agent of the same depth estimate. It cannot have had a parent of the same depth estimate before the cycle was created, as that would have set $\mathbb{I}^{[j]}_{\text{par-less}} = \text{false}$ and prevented i from connecting to j. This argument can be carried all the way around the cycle, C, and we can conclude that for each agent $k \in C$, k attached to a new parent at the time the cycle was formed. Here we invoke the UID-based tie-breaking scheme of step `6:`, and note that some agent i in this cycle must have been greater than either of its neighbors in C, which explicitly triggers the execution of step `7:`, preventing i from attaching to $g_{\text{p}}^{[i]}$ and thus preventing the formation of a cycle. □

The above result is the basis for our study in Section 5.3 of the repair properties of the CM Algorithm. The following result is a direct consequence.

Corollary 5.2.3. *Since no node i with $n^{[i]}_{root} = 0$ prefers to attach to a node j with $n^{[j]}_{root} \neq 0$, if no edges of the form $(i, f_p^{[i]})$ or $(i, g_p^{[i]})$ are removed, the graph of $(i, f_p^{[i]})$ for i having $n^{[i]}_{root} = 0$ remains connected and never decreases in size.*

5.3 Tree repair properties

We begin this section by showing that it is impossible to make a repair algorithm which allows links to break while at the same time handling all possible agent failures. This justifies our ensuing study of the link repair properties of CM Algorithm.

We divide our analysis in three parts. In Section 5.3.1, we show that repair works under the CM Algorithm if the underlying network remains connected. The agents' mobility might break the connectivity of the underlying network. In Section 5.3.2, we introduce the *restricted graph on k*, for $k \in \mathbb{Z}_n$, which is a key notion for establishing the repair properties in the dynamic case. Section 5.3.3 establishes the repair properties of CM Algorithm. Theorem 5.3.8 shows that any network config-

uration belonging to a wide class of configurations result in the network remaining connected for all time. Corollary 5.3.9 and Theorem 5.3.11 show that this class of configurations represent encompass situations where the network starts from an initially disconnected tree and becomes connected $O(n)$ rounds after the initial failure.

Let us begin with the impossibility result mentioned above.

Theorem 5.3.1. *There is no distributed repair algorithm which can allow links to break and, at the same time, recover from all possible underlying hardware failures which leave the communication graph connected.*

Proof. Consider an edge between agents i and j which can safely be broken at time t in the absence of underlying hardware failures. Let there be a cut of the graph which includes (i, j) and no other links sharing nodes i and j. The remaining edges in the cut could be severed by hardware failure immediately after the algorithm informs i and j that their link is safe to break, and the information would require at least one round to reach i and j. □

5.3.1 Repair properties when the underlying graph is connected

Here, we study the repair properties when the underlying graph remains connected.

Lemma 5.3.2. *For any node $i \in \mathbb{Z}_n$, if $n_{root}^{[i]}$ stays constant for $k \leq n_{root}^{[i]}(n+1)$ iterations, and no link failures happen in the intervening time, then, at the end of these iterations, $dpth_{est}^{[i]} \geq k$.*

Proof. We induct on the number of iterations. At iteration 0 each node, i, satisfies $\mathrm{dpth}_{\mathrm{est}}^{[i]} \geq 0$. By the update rule, $\mathrm{dpth}_{\mathrm{est}}^{[i]}$ at iteration k gets 1 more than the depth estimate of $f_{\mathrm{p}}^{[i]}$ at iteration $k-1$. Since no node ever attaches to a parent with greater $n_{\mathrm{root}}^{[\cdot]}$, $n_{\mathrm{root}}^{[f_{\mathrm{p}}^{[i]}]}$ must have been greater than or equal to

$n_{\text{root}}^{[i]}$ for all $k-1$ previous iterations, and, by induction, $\text{dpth}_{\text{est}}^{[f_{\text{p}}^{[i]}]}$ must have been at least $k-1$. This makes $\text{dpth}_{\text{est}}^{[i]}$ at least $1+(k-1)=k$ at the end of the kth iteration. □

The next result shows that this algorithm can repair breaks in the spanning tree whenever the underlying graph remains connected.

Theorem 5.3.3. *Let the communication graph of some component, K, of the network remain connected and not connect to any other components. Assume the (not necessarily connected) constraint tree is such that the only cycles are self-loops (i.e., $i = f_p^{[i]}$). Let id_K be the smallest UID of any node in the connected component, and denote the number of agents in the component by n_K. Within $id_K(n+1)+n+n_K$ iterations of* CM Algorithm, *every node, $i \in K$, will have $n_{root}^{[i]} = id_K$.*

Proof. By Lemma 5.3.2 within $(\text{id}_K+1)(n+1)$ rounds each node i with $n_{\text{root}}^{[i]} > \text{id}_K$ will have $\text{dpth}_{\text{est}}^{[i]} \geq (\text{id}_K+1)(n+1) > \text{id}_K(n+1)$. Since the underlying graph remains connected at all times, there is some node i having $n_{\text{root}}^{[i]} \neq \text{id}_K$ with an edge (in the underlying graph) between itself and some j having $n_{\text{root}}^{[j]} = \text{id}_K$. Agent i will prefer to attach to nodes $\{j \in \mathbb{Z}_n \ : \ n_{\text{root}}^{[j]} = \text{id}_K\}$ over all other nodes (the preference function is constrained so that i prefers to attach to nodes with smaller $n_{\text{root}}^{[\cdot]}$, and the smallest $n_{\text{root}}^{[\cdot]}$ available is id_K). Each of these nodes has $\text{dpth}_{\text{est}}^{[j]} \leq \text{id}_K(n+1)+n$, so the attach is allowed. This can happen for at most n iterations before every node i satisfies $n_{\text{root}}^{[i]} = \text{id}_K$. □

The above analysis does not work if the underlying graph does not remain connected. Disconnection can occur as a result of graph dynamics induced by robot motion. In the next section, we introduce a useful notion to analyze this more complex situation.

5.3.2 *Restricted graph on* k and its properties

Here, we show that *motion compatible with* CM algorithms enforce connectivity in situations in which the constraint tree is not fully connected. To do so, we will find it useful to introduce the following notion.

Definition 5.3.4. *The* restricted graph on k, *denoted* $G_{restr(k)}$, *is the graph consisting of all nodes* i *having* $n_{root}^{[i]} \leq k$ *and edges* (i, j), *where either* $j = f_p^{[i]}$, $i = f_p^{[j]}$, *or* $n_{root}^{[i]} \neq n_{root}^{[j]}$.

Next, proceed to show that $n_{\text{root}}^{[\cdot]}$ remains monotonically non-increasing along paths from parent to child whenever the network starts in a state with $n_{\text{root}}^{[\cdot]}$ monotonically non-increasing along paths from parent to child.

Lemma 5.3.5. *If no edge of the form* $(i, f_p^{[i]})$ *breaks,* CM Algorithm *maintains the invariant that* $n_{root}^{[\cdot]}$ *is monotonically non-increasing along edges from child to parent. As a corollary, the value of* $n_{root}^{[i]}$ *for any given* i *is monotonically non-increasing over time, given the first invariant (that* $n_{root}^{[\cdot]}$ *is monotonically non-increasing along edges from child to parent).*

Proof. $n_{\text{root}}^{[i]}$ can only change by taking on $n_{\text{root}}^{[f_{\text{p}}^{[i]}]}$, which does not affect the invariant. The value of $n_{\text{root}}^{[\cdot]}$ along the path from child to parent can change when a node re-attaches, but each node can only attach to nodes with $n_{\text{root}}^{[\cdot]}$ less than or equal to its own value of $n_{\text{root}}^{[\cdot]}$. □

We next show that any network converges, in a finite number of rounds, to a configuration where the property required in Lemma 5.3.5 is satisfied.

Lemma 5.3.6. *Given any starting condition, any* motion compatible with CM *algorithm guarantees that the* CM *algorithm within* n *rounds, each agent* i *will have the property that* $n_{root}^{[i]} \geq n_{root}^{[f_p^{[i]}]}$.

Proof. Let $S = \{i \in \mathbb{Z}_n \ : \ n_{\text{root}}^{[f_{\text{p}}^{[i]}]} > n_{\text{root}}^{[i]}\}$ and $S_> = \{j \in \mathbb{Z}_n \ : \ \exists i \in S, n_{\text{root}}^{[j]} \geq n_{\text{root}}^{[i]}\}$. Note that, since each $i \in S$ satisfies that $n_{\text{root}}^{[i]} \geq n_{\text{root}}^{[j]}$ for some $j \in S$ (namely $j = i$), we deduce that $S \subset S_>$. We argue that $|S_>|$ decreases by at least one every round. No node, k, will increase $n_{\text{root}}^{[k]}$ unless its parent connection is severed (prohibited by *motion compatible with* CM) or its parent had previously satisfied $n_{\text{root}}^{[f_{\text{p}}^{[k]}]} > n_{\text{root}}^{[k]}$, thus putting $k \in S_>$. Thus no node gets added to $S_>$. Let $l = \text{argmax}_{i \in S}\{n_{\text{root}}^{[f_{\text{p}}^{[i]}]}\}$ and note that $n_{\text{root}}^{[f_{\text{p}}^{[l]}]} > n_{\text{root}}^{[i]}$ for all $i \in S$. Since the update rule, $n_{\text{root}}^{[l]} \leftarrow n_{\text{root}}^{[f_{\text{p}}^{[l]}]}$ applies to l, at least one node must be removed from $S_>$ every round. When $|S_>| = 0$, we conclude that $S = \emptyset$. □

The next result is the crux of our connectivity argument for partially disconnected constraint trees. Proposition 5.3.7 forms the basis for our arguments linking the repair properties of CM Algorithm to the guarantees provided by any *motion compatible with* CM algorithm.

Proposition 5.3.7. *Given a network which starts with the invariant from Lemma 5.3.5, the following property is also an invariant of the* CM Algorithm *when combined with any* motion compatible with CM *algorithm: for any two robots i and j having $n_{root}^{[i]} = n_{root}^{[j]} = k$, there is a path between i and j in $G_{restr(k)}$.*

Proof. We will proceed by induction on k. For $k = 0$, the result follows from Corollary 5.2.3. Assume the result holds for $\kappa < k$. Let the property hold for k on round $t-1$. To show it holds for k on round t, consider the ways in which an edge can be removed from $G_{\text{restr}(k)}$ on round t:

- An edge, (i, j), in $G_{\text{restr}(k)}$ having $j \neq f_{\text{p}}^{[i]}$ and $i \neq f_{\text{p}}^{[j]}$ can disappear when $n_{\text{root}}^{[i]}$ becomes equal to $n_{\text{root}}^{[j]}$.

In this case, either $n_{\text{root}}^{[i]}$ or $n_{\text{root}}^{[j]}$ must have been less than k at round $t-1$. By Lemma 5.3.5, the value of $n_{\text{root}}^{[i]}$ and $n_{\text{root}}^{[j]}$ can only decrease, making each of these less than k at round t. By the induction

hypothesis, there must be a path between the two at round t in $G_{\text{restr}(n_{\text{root}}^{[i]})}$ which is also a path in $G_{\text{restr}(k)}$.

- Node i, having $f_{\text{p}}^{[i]} = h$ at round $t-1$, can change $f_{\text{p}}^{[i]}$.

    (i) to a new node, j, having $n_{\text{root}}^{[j]} < k$.

In this case, either $n_{\text{root}}^{[h]} \neq n_{\text{root}}^{[i]}$, leaving the two connected, or $n_{\text{root}}^{[h]} = n_{\text{root}}^{[i]}$, but since $n_{\text{root}}^{[i]}$ is the value $n_{\text{root}}^{[j]}$ had at round $t-1$, $n_{\text{root}}^{[i]}$ and $n_{\text{root}}^{[j]}$ must both be less than k, leaving the two connected, by the induction hypothesis; or

    (ii) to a new node, j, having $n_{\text{root}}^{[j]} = k$.

If $n_{\text{root}}^{[h]} \neq k$ at round t, i and h are still connected in $G_{\text{restr}(k)}$. Otherwise, since $n_{\text{root}}^{[h]}$ never decreases, we have $n_{\text{root}}^{[h]} = n_{\text{root}}^{[j]} = n_{\text{root}}^{[i]} = k$ at round $t-1$ meaning all three of i, j, h had paths between one another in $G_{\text{restr}(k)}$ at round $t-1$. We show that this sort of attach operation does not break the property we are trying to establish by ordering the attach and update operations in the following way. First we perform, in (an arbitrary) sequence, all parent re-attach operations from an old parent to a new parent with $n_{\text{root}}^{[\cdot]} = k$. Since this operation does not change the value of $n_{\text{root}}^{[\cdot]}$ for the node switching parents, we do not worry about the $n_{\text{root}}^{[\cdot]}$ update for this kind of event. After showing that none of these operations individually break the connectivity among agents with $n_{\text{root}}^{[\cdot]} = k$ in $G_{\text{restr}(k)}$, we then proceed with the remainder of attach and update operations in the synchronous order in which they occur in the algorithm, relying on the correctness properties already established to show these operations correct. First, to show none of the $n_{\text{root}}^{[i]} = n_{\text{root}}^{[j]} = n_{\text{root}}^{[h]} = k$ operations break connectivity we note, as shown above, that at round $t-1$ there is a path between any pair of i, j and h before the attach. After changing $f_{\text{p}}^{[i]}$ from h to j, there is still a path from h to j in $G_{\text{restr}(k)}$, meaning there are paths between i and j (trivially) and between i and h (through path composition) after the event.

Since all three affected nodes are still interconnected after the event, the connectivity of any pair of nodes in $G_{\text{restr}(k)}$ is not affected by the re-attach, and we can use the same line of reasoning for the next re-attach in our sequence.

Finally we note that when a new node, i, sets $n_{\text{root}}^{[i]} = k$ for the first time, it is entering $G_{\text{restr}(k)}$ for the first time. It is doing so by attaching (or staying attached) to a node, j, which had $n_{\text{root}}^{[j]} = k$ at round $t-1$, and thus j was connected to all nodes having $n_{\text{root}}^{[\cdot]} = k$ at round $t-1$. If we order the sequence of updates again, putting these attaches and $n_{\text{root}}^{[\cdot]}$ updates before the ones we have already considered, we show that they also do not break the connectivity among nodes with $n_{\text{root}}^{[\cdot]} = k$ in $G_{\text{restr}(k)}$. □

5.3.3 Repair properties under dynamic graph conditions

In this section, we study the repair properties of the CM Algorithm when the connectivity of the underlying graph is evolving. We begin by showing that the algorithm maintains the connectivity of any network which starts with $G_{\text{restr}(k)}$ connected for $k \in \{0, \ldots, n\}$, provided $n_{\text{root}}^{[\cdot]}$ is initially monotonically non-increasing along links from child to parent.

Theorem 5.3.8. *Consider the composition of* CM Algorithm *with a* motion compatible with CM *algorithm. Assume the network starts in a configuration where*

(i) the restricted graph on n *is connected,*

(ii) any two agents, $i, j \in \mathbb{Z}_n$, having $n_{root}^{[i]} = n_{root}^{[j]} = k$ are connected in $G_{restr(k)}$, and

(iii) along each path from child node to parent node, the value of $n_{root}^{[\cdot]}$ is monotonically non-increasing.

Then, the network remains connected for all time.

Proof. Consider the links of the graph between rounds $t-1$ and t. Each link in the graph is either between a pair of nodes with the same value of $n_{\text{root}}^{[\cdot]}$ or between a pair of nodes with different values of $n_{\text{root}}^{[\cdot]}$. By Proposition 5.3.7, any pair of nodes having $n_{\text{root}}^{[\cdot]} = k$ are connected in $G_{\text{restr}(k)}$ (and thus are connected in $G_{\text{restr}(n)}$) at all times for any such starting configuration. Any pair of nodes, i, j, having $n_{\text{root}}^{[i]} \neq n_{\text{root}}^{[j]}$ will be preserved during the motion phase (by properties of *motion compatible with* CM). During the computation phase, either $n_{\text{root}}^{[i]}$ and $n_{\text{root}}^{[j]}$ will become the same, which guarantees a path exists between the two in $G_{\text{restr}(n_{\text{root}}^{[i]})}$ and thus in $G_{\text{restr}(n)}$, or $n_{\text{root}}^{[i]}$ and $n_{\text{root}}^{[j]}$ will remain different, in which case the edge will remain in $G_{\text{restr}(n)}$. Since any link , (i, j), either remains in $G_{\text{restr}(n)}$ or disappears, but provably maintains a path between i and j in $G_{\text{restr}(n)}$, any nodes starting connected in $G_{\text{restr}(n)}$ will remain so for all time. □

Showing that the constraints guaranteed by the notion of *motion compatible with* CM algorithm achieve $G_{\text{restr}(\cdot)}$-connectivity is not sufficient to show that failed links are repaired along the execution. We now proceed to show that connectivity of the underlying graph holds under a more flexible set of conditions than those proposed in Theorem 5.3.8.

First, let us show that the constraints imposed by the notion of *motion compatible with* CM algorithm suffice to guarantee connectivity in the case where we use the repair properties of CM Algorithm to build our initial spanning tree from the default start state of the algorithm.

Corollary 5.3.9. *When* CM Algorithm *is coupled with a* motion compatible with CM *algorithm the following holds: if the network starts in a state where ever node i has $f_p^{[i]} = n_{root}^{[i]} = i$ and the communication graph is connected, the swarm will stay connected at all time.*

As a prelude to the main result, we present Lemma 5.3.10, which shows that n rounds is sufficient time to satisfy the pre-conditions of Theorem 5.3.8.

Lemma 5.3.10. *Use n_k to denote the number of rounds it takes for each node, i, having $n_{root}^{[i]} = k$ to be connected to k in $G_{restr(k)}$ given that the network starts with a connected underlying graph. This number obeys the following relationship : $n_k \leq n - n_{k-1}$ (i.e. within at most $n - n_{k-1}$ rounds, each node, i, having $n_{root}^{[i]} = k$ will be connected to k in $G_{restr(k)}$, leaving at most $n - n_k$ nodes with $n_{root}^{[\cdot]} > k$).*

Proof. From Lemma 5.3.6, note that $n_{\text{root}}^{[i]} \geq n_{\text{root}}^{[f_{\text{P}}^{[i]}]}$ holds for each agent i. We will proceed by induction on k. As in Proposition 5.3.7, our base case, with $k = 0$ follows from Corollary 5.2.3 of Theorem 5.2.2. Assume our hypothesis holds for all $\kappa < k$. So long as there are nodes with $n_{\text{root}}^{[\cdot]} = k$ not connected to k in $G_{\text{restr}(k)}$, at least one such node must have a parent with $n_{\text{root}}^{[\cdot]} < k$ (any such node must have a parent other than itself, otherwise it would have UID equal to k). So for each round during which there are such nodes, one such node must permanently get $n_{\text{root}}^{[\cdot]} < k$. No node having $n_{\text{root}}^{[\cdot]} = k$ which is connected to k in $G_{\text{restr}(k)}$ after round $n(k-1)$ ever becomes disconnected because the only way for edges in $G_{\text{restr}(k)}$ to become severed are

- Through re-attach: however, this guarantees the child node gets a new root. Unless the parent gets a new root as well, the child remains attached to the parent.
- An edge between i and j in $G_{\text{restr}(k)}$ disappears when i and j get the same value of $n_{\text{root}}^{[\cdot]}$, or i detaches from its old parent, j, and both i and j get the same value of $n_{\text{root}}^{[\cdot]}$. Since i and j were already in $G_{\text{restr}(k)}$, they must get new $n_{\text{root}}^{[\cdot]}$ values less than k, but by induction, must therefore be connected in $G_{\text{restr}(n_{\text{root}}^{[i]})}$.

Since there are at most $n - n_{k-1}$ nodes which could possibly have $n_{\text{root}}^{[\cdot]} > k - 1$, after at most $n - n_{k-1}$ more rounds, no nodes having $n_{\text{root}}^{[\cdot]} = k$ remain unless they are connected to k in $G_{\text{restr}(k)}$. For each round between round n_{k-1} and round n_k, where n_k is the minimum time at which each node having $n_{\text{root}}^{[\cdot]} = k$ is connected to k in $G_{\text{restr}(k)}$, at least one node per round goes from $n_{\text{root}}^{[\cdot]} = k$ to

$n_{\text{root}}^{[\cdot]} < k$, leaving at most $n - n_k$ nodes with $n_{\text{root}}^{[\cdot]} > k$. □

Note that the above result implies $0 \leq n_k \leq n$ for each n_k. Finally, the next result shows that after $2n$ rounds with no underlying hardware failures, once the communication graph becomes connected, it will remain connected for all time.

Theorem 5.3.11. *If the graph starts in any initial state (even if the underlying graph is disconnected), and the evolution of the network follows the constraints imposed by the notion of* motion compatible with CM *algorithm for* $2n$ *rounds, then, if the underlying graph becomes connected again, it will stay connected for all time, thus building a connected constraint tree.*

Proof. After n rounds, the value of $n_{\text{root}}^{[\cdot]}$ along links from children to parents is monotonically non-increasing by Lemma 5.3.6. By Lemma 5.3.10, which shows that another n rounds suffices to satisfy the preconditions of Theorem 5.3.8, any time the graph becomes connected, it will remain connected for all time. □

5.4 Reachability properties

Here we examine the reachability properties of the CM Algorithm. Roughly speaking, by reachability we mean that the algorithm is capable of switching between any two given constrained trees. After providing a formal definition, we show that the CM Algorithm combined with the Null Depth Increment Algorithm does not satisfy reachability. This motivates the introduction of a new *depth compatible with* CM algorithm, termed Cycle-Detecting Depth Increment Algorithm. After characterizing the execution of the CM Algorithm combined with Cycle-Detecting Depth Increment Algorithm, we are able to show that the resulting input-output control and communication law satisfies reachability.

We begin by defining what it means for one tree to be reachable from another. Note that the evolution of the constraint tree depends not only on CM Algorithm, but also on the algorithms it is coupled with. We take this into account in our following definition of reachability.

Definition 5.4.1 (Reachable trees)**.** *We say a constraint tree T_2 is reachable from a constraint tree T_1 with a sequence of underlying graphs $\{\mathcal{G}(t)\}_{t\in\mathbb{N}}$ under* CM Algorithm *coupled with a* depth compatible with CM *algorithm A if the following conditions hold for any initial allocation of $dpth_{est}^{[i]}, i \in \mathbb{Z}_n$ satisfying $dpth_{est}^{[0]} = 0$ and $dpth_{est}^{[i]} \geq dpth_{est}^{[f_p^{[i]}]}$ for all $i \in \mathbb{Z}_n$.*

- *i is allowed to set $f_p^{[i]} = j$ or $g_p^{[i]} = j$ at round t only if $(i, j) \in \mathcal{E}(\mathcal{G}(t))$,*
- *T_1 and T_2 are subgraphs of each $\mathcal{G}(t)$,*
- *There exists a set of link preferences (values of $R_{pref}^{[i]}, i \in \mathbb{Z}_n$) such that, under* CM Algorithm *coupled with A, the constraint tree will eventually settle on T_2 if it starts in T_1,*

Next, we define what is an algorithm that satisfies the reachability property.

Definition 5.4.2. *A* depth compatible with CM *algorithm A satisfies the* reachability property *if, for every pair of trees T_1 and T_2, and any sequence of graphs $\mathcal{G}(t)$, having T_1 and T_2 subgraphs of $\mathcal{G}(t)$ for all t, T_2 is reachable from T_1 with underlying graphs $\{\mathcal{G}(t)\}_{t\in\mathbb{N}}$ under* CM Algorithm *combined with A.*

Consequently, an algorithm which satisfies the reachability property can drive the constraint tree to any desired tree given a suitable preference function. It turns out that Null Depth Increment Algorithm does not satisfy the reachability property, as we show next.

Theorem 5.4.3. *The* CM Algorithm *coupled with* Null Depth Increment Algorithm *does not have the reachability property.*

Proof. Let G be the cycle graph on n vertices, with $(i, (i+1) \bmod n) \in \mathcal{E}(G)$ for every $i \in \mathbb{Z}_n$. Let T_1 have $f_{\mathrm{p}}^{[n-1]} = 0$ and $f_{\mathrm{p}}^{[i]} = i - 1$ for all i other than 0 and $n-1$. Let T_2 have $f_{\mathrm{p}}^{[1]} = 0$ and $f_{\mathrm{p}}^{[i]} = (i+1) \bmod n$ for all i other than 0 and 1. Let the initial depth estimates, $\mathrm{dpth}_{\mathrm{est}}^{[i]}$, be equal to the exact depth of i in the tree T_1. Node $n-1$ cannot attach to anything other than 0 until node $n-2$ has a depth $\mathrm{dpth}_{\mathrm{est}}^{[n-2]} \leq 1$. But the only nodes $n-2$ can attach to are $n-3$ and $n-1$. If it attaches to $n-1$ it will get a depth estimate $\mathrm{dpth}_{\mathrm{est}}^{[n-2]} = 2$. Since $n-3$ cannot attach to the root, it will always have a depth estimate $\mathrm{dpth}_{\mathrm{est}}^{[n-3]} \geq 2$, so if $n-2$ attaches to $n-3$ it will have $\mathrm{dpth}_{\mathrm{est}}^{[n-2]} \geq 3 > 1$ thus preventing node $n-1$ from ever attaching to it. □

5.4.1 Cycle-Detecting Depth Increment Algorithm

Given the negative result in Theorem 5.4.3, here we introduce a new *depth compatible with* CM algorithm, termed Cycle-Detecting Depth Increment Algorithm. Informally, this algorithm performs two operations, described as follows.

> *[Informal description:]* Each robot stores a "start number", a "number of descendants" and a "mapping from child UID to child start number." At each round, in addition to the tree constraint information and its "number of descendants", each node sends the following information to each neighbor. If the neighbor is a child, it sends the appropriate entry in its mapping, or, if the child is not in the mapping, it sends its own "start number." If the neighbor is not a child, it sends its own "start number." With the messages received, each node updates its numbers in the following way. Its "number of descendants" is the sum of the "number of descendants" information received from each child, plus one (for itself). Its "start number" is the one received from its parent. For each child it receives a message from, it adds an entry to its map that is indexed by that child's UID and has a value of "the sum, over all children with lesser UID, of the number of descendants of those children, plus one plus its own start number."

The formal description of Cycle-Detecting Depth Increment Algorithm as an input-output control and communication law consists of the sets

(i) $L = (\mathbb{Z}_n \cup \{\emptyset\}) \times \mathbb{Z}$ (further explanation of L will be provided when we introduce msg);

(ii) $W = (\mathbb{Z}_n \cup \{\emptyset\}) \times \mathbb{Z}_2 \times \mathbb{Z} \times \mathbb{Z} \times \mathfrak{F}(\mathbb{Z}_n; \mathbb{Z})$, are sets of values of logic variables

$w^{[i]} = (n^{[i]}_{\text{incr-sgnl}}, \mathbb{I}^{[i]}_{\text{incr-dep}}, n^{[i]}_{\text{start}}, n^{[i]}_{\text{num-desc}}, f^{[i]}_{\text{start}})$, $i \in I$,

where $n^{[i]}_{\text{incr-sgnl}}$ is used to break cycles, while $n^{[i]}_{\text{start}}$, $n^{[i]}_{\text{num-desc}}$ and $f^{[i]}_{\text{start}}$ are used to detect descendants;

(iii) $W_0^{[i]} = \{(\emptyset, \text{false}, 0, 0, f^{[i]}_{\text{start}})\} \subset W$, with $f^{[i]}_{\text{start}} \equiv 0$;

(iv) $W_{\text{in}} = \mathbb{Z}_n \cup \{\emptyset\}$, ${w_{\text{in}}}^{[i]} = n^{[i]}_{\text{dep-targ}}$, $i \in I$;

(v) ${W_{\text{in},0}}^{[i]} = \{\emptyset\}$;

(vi) $W_{\text{out}} = \mathbb{Z}_2 \times (\mathbb{Z}_n \cup \{\emptyset\}) \times \mathfrak{F}(\mathbb{Z}_n; \{\text{true}, \text{false}\})$, are sets of values of output logic variables ${w_{\text{out}}}^{[i]} = (\mathbb{I}^{[i]}_{\text{incr-dep}}, n^{[i]}_{\text{incr-sgnl}}, f^{[i]}_{\text{allow}})$, $i \in I$;

and of the maps:

(i) $\text{msg}(x, w, w_{\text{in}}, i) = \begin{cases} (n^{[\text{id}]}_{\text{incr-sgnl}}, f^{[\text{id}]}_{\text{start}}(i)) & f^{[i]}_{\text{p}} = \text{id} \\ (n^{[\text{id}]}_{\text{incr-sgnl}}, n^{[\text{id}]}_{\text{num-desc}}) & f^{[i]}_{\text{p}} \neq \text{id} \end{cases}$; Each robot, id sends $n^{[\text{id}]}_{\text{incr-sgnl}}$ to each of its neighbors, and, to each neighbor j, sends either $f^{[\text{id}]}_{\text{start}}(j)$ to j if $f^{[j]}_{\text{p}} = \text{id}$ or sends $n^{[\text{id}]}_{\text{num-desc}}$ to j otherwise, with the intent that $n^{[j]}_{\text{start}}$ will be set to $f^{[\text{id}]}_{\text{start}}(j)$;

(ii) $\text{stf}_{\text{io}-\text{slf}}$ as described in Table 5.1 ;

(iii) $\text{stf}_{\text{io-out}}(w_{\text{in}}, w, \{l_i\}_{i \in \mathcal{N}(\text{id})}) = (\mathbb{I}^{[\text{id}]}_{\text{incr-dep}}, n^{[i]}_{\text{incr-sgnl}}, f^{[\text{id}]}_{\text{allow}})$, where

$f^{[\text{id}]}_{\text{allow}}(j) = \begin{cases} \text{true} & ([n^{[\text{id}]}_{\text{start}}, n^{[\text{id}]}_{\text{start}} + n^{[\text{id}]}_{\text{num-desc}}] \cap [n^{[j]}_{\text{start}}, n^{[j]}_{\text{start}} + n^{[j]}_{\text{num-desc}}] = \emptyset) \\ \text{false} & \text{otherwise} \end{cases}$;

(iv) $\text{ctl}(x_{t_\ell}, x, w, w_{\text{in}}) = 0$.

Before tackling a detailed analysis of the properties of the Cycle-Detecting Depth Increment Algorithm, we show next that the variables $n^{[i]}_{\text{num-desc}}$ and $n^{[i]}_{\text{start}}$ of the processor state can be used to correctly answer queries of the form "is i a descendant of j?"

function $\mathrm{stf}_{\mathrm{io-slf}}(n^{[\mathrm{id}]}_{\text{dep-targ}}, (n^{[\mathrm{id}]}_{\text{incr-sgnl}}, \mathbb{I}^{[\mathrm{id}]}_{\text{incr-dep}}), l)$

$n^{[\mathrm{id}]}_{\text{num-desc}} \leftarrow \sum_{j \in \{j \in \mathcal{N}(i) \,:\, f_{\mathrm{p}}^{[j]} = i\}} (1 + n^{[j]}_{\text{num-desc}})$

$n^{[\mathrm{id}]}_{\text{start}} \leftarrow n^{[f_{\mathrm{p}}^{[i]}]}_{\text{start}}$

$f^{[\mathrm{id}]}_{\text{start}}(j) \leftarrow n^{[\mathrm{id}]}_{\text{start}} + \sum_{\{k \in S^{[\mathrm{id}]}_{\text{desc}} \,:\, k < j\}} n^{[k]}_{\text{num-desc}}$ for each $j \in \mathcal{N}(\mathrm{id})$

if $n^{[f_{\mathrm{p}}^{[\mathrm{id}]}]}_{\text{incr-sgnl}} \neq \emptyset$ **then**

  Set $n^{[\mathrm{id}]}_{\text{incr-sgnl}} \leftarrow n^{[f_{\mathrm{p}}^{[\mathrm{id}]}]}_{\text{incr-sgnl}}$; Set $\mathbb{I}^{[\mathrm{id}]}_{\text{incr-dep}} \leftarrow$ false

else if $n^{[\mathrm{id}]}_{\text{dep-targ}} \neq \emptyset$ and $n^{[n^{[\mathrm{id}]}_{\text{dep-targ}}]}_{\text{incr-sgnl}} \neq \emptyset$ **then**

  if $n^{[n^{[\mathrm{id}]}_{\text{dep-targ}}]}_{\text{incr-sgnl}} < n^{[\mathrm{id}]}_{\text{incr-sgnl}}$ or $n^{[\mathrm{id}]}_{\text{incr-sgnl}} = \emptyset$ **then**

    Set $n^{[\mathrm{id}]}_{\text{incr-sgnl}} \leftarrow n^{[n^{[\mathrm{id}]}_{\text{dep-targ}}]}_{\text{incr-sgnl}}$

  else if $n^{[n^{[\mathrm{id}]}_{\text{dep-targ}}]}_{\text{incr-sgnl}} = \mathrm{id}$ **then**

    Set $\mathbb{I}^{[\mathrm{id}]}_{\text{incr-dep}} \leftarrow$ false

  end if

end if

return $(n^{[\mathrm{id}]}_{\text{incr-sgnl}}, \mathbb{I}^{[\mathrm{id}]}_{\text{incr-dep}})$

Table 5.1: $\mathrm{stf}_{\mathrm{io-slf}}$ for Cycle-Detecting Depth Increment Algorithm.

Lemma 5.4.4. *If the constraint tree T remains fixed for $2\,\mathrm{depth}(T)$ rounds, then each i satisfies*

(i) $n^{[i]}_{\textit{num-desc}}$ *is equal to the total number of descendants of* i*,*

(ii) $n^{[i]}_{\textit{start}} = \sum_{j <_{lex-T} i} 1$*,*

(iii) if i *is an ancestor of* j*, eventually* $n^{[i]}_{\textit{start}} \leq n^{[j]}_{\textit{start}}$ *and* $n^{[i]}_{\textit{start}} + n^{[i]}_{\textit{num-desc}} \geq n^{[j]}_{\textit{start}} + n^{[j]}_{\textit{num-desc}}$ *will hold, and*

(iv) if i *is not an ancestor of* j*, either* $n^{[i]}_{\textit{start}} > n^{[j]}_{\textit{start}} + n^{[j]}_{\textit{num-desc}}$ *or* $n^{[j]}_{\textit{start}} > n^{[i]}_{\textit{start}} + n^{[i]}_{\textit{num-desc}}$ *will hold.*

Proof. Each node, i, has the proper value of $n^{[i]}_{\text{num-desc}}$ one round after the last of its children have the proper $n^{[\cdot]}_{\text{num-desc}}$. This takes at most $\mathrm{depth}(T)$ rounds, thus showing *(i)*. To show *(ii)*, each node has the proper value of $n^{[i]}_{\text{start}}$ one round after its siblings each have the proper value of $n^{[\cdot]}_{\text{num-desc}}$ and its parent, $f_{\mathrm{p}}^{[i]}$, has the proper value of $n^{[f_{\mathrm{p}}^{[i]}]}_{\text{start}}$. This takes at most $\mathrm{depth}(D)$ additional rounds after all nodes satisfy

(i). Regarding fact *(iii)*, after $2\,\mathrm{depth}(T)$ rounds, for each node $m \in \mathbb{Z}_n$, $n^{[m]}_{\text{num-desc}}$ will be the total number of descendants of m and $n^{[\cdot]}_{\text{start}}$ will be $\sum_{k<_{\text{lex}-T}m} 1$. If i is not an ancestor of j, without loss of generality let $j > i$. Let k be the nearest common ancestor of i and j. Let k_i and k_j be the children of k having i and j as descendants respectively. At some point k_i will get some $n^{[k_i]}_{\text{start}}$ and k_j will get $n^{[k_j]}_{\text{start}} \geq n^{[k_i]}_{\text{start}} + n^{[k_i]}_{\text{num-desc}}$ thus showing *(iii)*. Regarding fact *(iv)*, if i is an ancestor of j, for every link, (k_1, k_2), between i and j, k_2 will get some number between $n^{[k_1]}_{\text{start}}$ and $n^{[k_1]}_{\text{start}} + n^{[k_1]}_{\text{num-desc}} - n^{[k_2]}_{\text{num-desc}}$ for $n^{[k_2]}_{\text{start}}$. Thus, at every step along the way, $n^{[k_1]}_{\text{start}} \leq n^{[k_2]}_{\text{start}} \leq n^{[k_2]}_{\text{start}} + n^{[k_2]}_{\text{num-desc}} \leq n^{[k_1]}_{\text{start}} + n^{[k_1]}_{\text{num-desc}}$. By inducting on the number of links from i, this gives *(iv)*. □

5.4.2 Properties of the evolutions under Cycle-Detecting Depth Increment Algorithm

Here we gather some properties of the executions of CM Algorithm combined with Cycle-Detecting Depth Increment Algorithm. These results will later be instrumental to establish the reachability properties satisfied by this combination. We begin by introducing several auxiliary graphs.

Definition 5.4.5. *Consider the following subgraphs of the underlying graph...*

- *The* increase depth graph *is the graph containing all edges of the form* $(i, n^{[i]}_{dep\text{-}targ})$ *for* $n^{[i]}_{dep\text{-}targ} \neq \emptyset$ *and all edges of the form* $(i, f^{[i]}_p)$ *where* $n^{[i]}_{incr\text{-}sgnl} \neq \emptyset$ *and* $n^{[f^{[i]}_p]}_{incr\text{-}sgnl} \neq \emptyset$*;*

- *The* lowest subgraph (in the increase depth graph) *consists of all edges* $(i, n^{[i]}_{dep\text{-}targ})$ *where no ancestor,* j*, of* i *has* $n^{[j]}_{dep\text{-}targ} \neq \emptyset$ *and all edges of the form* $(i, f^{[i]}_p)$ *where some ancestor of* i *is also in the lowest subgraph;*

- *The* signal graph on i *is the graph containing all edges of the form* (j, k) *where* $n^{[j]}_{incr\text{-}sgnl} = i$*,* $n^{[k]}_{incr\text{-}sgnl} = i$ *and either* $k = f^{[j]}_p$ *or* $k = n^{[j]}_{dep\text{-}targ}$*;*

Each of these graphs contain only those nodes adjacent to an edge in the respective graph.

Note that the increase depth graph and its lowest subgraph only change when the underlying graph, the constraint tree, or $n^{[i]}_{\text{dep-targ}}$ for some i change. The signal graph on i can change even when the aforementioned structures remain constant.

We proceed to show that whenever the set of agent preferences are such that each node in the network would prefer its parent in T_2 to its parent in the current tree, some node will change parents. We break this down by cases.

(i) The "lowest subgraph of the increase depth graph" contains no cycles.

(ii) The "lowest subgraph of the increase depth graph" contains at least one cycle.

Case (i) is treated in Lemma 5.4.6, which describes what happens when the set of preferences do not induce a cycle in the "lowest subgraph of the increase depth graph." Case (ii) is handled in Lemma 5.4.8.

Lemma 5.4.6. *If there is no cycle in the "lowest subgraph in the increase depth graph" then some node j in this graph will attach to $n^{[j]}_{dep\text{-}targ}$ within a finite number of time steps.*

Proof. If there is no such cycle, than some edge of the form $(i, n^{[i]}_{\text{dep-targ}})$ exists such that neither $n^{[i]}_{\text{dep-targ}}$ nor any of its ancestors has $n^{[\cdot]}_{\text{dep-targ}} \neq \emptyset$. Since i is in the lowest subgraph of the increase depth graph, each ancestor, j, of i has $n^{[j]}_{\text{dep-targ}} = \emptyset$. If this is the case, than $n^{[i]}_{\text{dep-targ}}$ will settle to a depth estimate, $\text{dpth}^{[n^{[i]}_{\text{dep-targ}}]}_{\text{est}}$, of its actual depth in the tree (less than $n+1$). i will never receive $n^{[\cdot]}_{\text{incr-sgnl}} = i$ and will keep increasing depth until it can attach to $n^{[i]}_{\text{dep-targ}}$ (at a value of, at most, $n+1$). □

If there is a cycle in the "lowest subgraph of the increase depth graph" the signaling mechanism of Cycle-Detecting Depth Increment Algorithm will detect it, as shown in Lemma 5.4.7.

Lemma 5.4.7. *If there is a cycle in the "lowest subgraph of the increase depth graph" then within a finite number of timesteps, there exists some k such that the "signal graph on k" contains that cycle.*

Proof. There are two cases in which a node can switch from $n^{[i]}_{\text{incr-sgnl}} = j$ to $n^{[i]}_{\text{incr-sgnl}} = k$. One is when its parent sends $n^{[f^{[i]}_{\text{p}}]}_{\text{incr-sgnl}} = k$. This does not happen for any i having $(i, n^{[i]}_{\text{dep-targ}})$ in the lowest subgraph of the increase depth graph. The other is when $n^{[n^{[i]}_{\text{dep-targ}}]}_{\text{incr-sgnl}} = k$ and $k < j$. The nodes i in any cycle of lowest subgraph having $(i, n^{[i]}_{\text{dep-targ}})$ in the lowest subgraph have a unique minimum UID, k. When some edge, $(i, f^{[i]}_{\text{p}})$ in the subgraph has $n^{[f^{[i]}_{\text{p}}]}_{\text{incr-sgnl}} = k$, in the next round, $n^{[i]}_{\text{incr-sgnl}}$ gets k. Likewise when some $(i, n^{[i]}_{\text{dep-targ}})$ in the subgraph gets $n^{[n^{[i]}_{\text{dep-targ}}]}_{\text{incr-sgnl}} = k$, in the next round, $n^{[i]}_{\text{incr-sgnl}} = k$. Since these edges describe every edge in the lowest subgraph of the increase depth graph, eventually all nodes in this cycle will have $n^{[\cdot]}_{\text{incr-sgnl}} = k$. □

This detection mechanism makes our proof for case (ii) above possible, as we show next.

Lemma 5.4.8. *If there is an edge, (i, j) in one of the cycles of the "lowest subgraph of the increase depth graph" where j is not an ancestor of i and vice versa, then within a finite number of time steps, some node, k, in this graph will attach to $n^{[k]}_{dep\text{-}targ}$.*

Proof. By Lemma 5.4.7, every edge of this cycle will get $n^{[\cdot]}_{\text{incr-sgnl}} = k$ for some k. Node k will reset, as it receives $n^{[n^{[k]}_{\text{dep-targ}}]}_{\text{incr-sgnl}} = k$, as will every descendant trying to attach to some node dependent on this cycle. Some node which is not a descendant of k must be trying to attach to a descendant of k, and will keep increasing depth while k's ancestors stay reset. □

Finally, Lemma 5.4.9 and Theorem 5.4.10 tie the two cases together.

Lemma 5.4.9. *If the graph and the constraint tree do not change, for $2\,\text{depth}(T)$ time steps, either the conditions for Lemma 5.4.8 or Lemma 5.4.6 will hold, or no node, i, will have $n^{[i]}_{dep\text{-}targ} \neq \emptyset$*

Proof. By Lemma 5.4.4, after $2\,\mathrm{depth}(T)$ time steps, no i and j where j is a descendant of i will have $f^{[i]}_{\text{allow}}(j) = \text{true}$. CM Algorithm will not allow $n^{[i]}_{\text{dep-targ}}$ to be j and thus any cycle in the lowest subgraph of the increase depth graph must contain at least one of the type of edge described in Lemma 5.4.8. □

The next result shows that as long as some node wishes to attach to a node which is not one of its descendants, the constraint tree will change.

Theorem 5.4.10. *If the underlying graph and the constraint tree do not change for* $2\,\mathrm{depth}(T)$ *rounds, and some node desires to attach to a node other than its descendants, one such node will eventually do so (causing the constraint tree to change).*

Proof. Either there is some node, i which wants to attach to a node j having $\mathrm{dpth}^{[j]}_{\text{est}} < \mathrm{dpth}^{[i]}_{\text{est}}$, or (by Lemma 5.4.9) the lowest subgraph of the increase depth graph contains no cycles, or the lowest subgraph of the increase depth graph contains at least one cycle. The first case is trivial, the second is handled by Lemma 5.4.6 and the third is handled by Lemma 5.4.8. □

5.4.3 Reachability analysis

We now approach the reachability problem. Our strategy is as follows: we give the network the simplest set of preferences that yield T_2 as the most desirable tree. We then show that this set of preferences yields T_2 given sufficient time. Theorem 5.4.12 is the culmination of this line of reasoning. Its proof classifies the edges of T_2 into (i) those which are in the current tree, (ii) those which will be in the current tree within a finite amount of time, and (iii) those which cannot currently be added to the current tree.

To show that the existence of edges in the third case requires the existence of edges in the second, we prove the following result.

Lemma 5.4.11. *Consider what happens when we start with a valid tree, T, and begin replacing edges in T with edges from nodes back to their ancestors in T. Any sequence of such operations will yield a graph which is not connected.*

Proof. The first such edge we replace will disconnected all the descendants of some node, i, from the root. Any replacement involving one of the now disconnected nodes changing its parent will not connect these nodes back to the root, since the disconnected node is attaching back to one of its ancestors. Any replacement involving a currently connected node will also not connect these nodes to the root, as the currently connected node loses its path to the root when the old edge from itself to its parent is removed. □

We are now ready to characterize the reachability property holds for the CM Algorithm coupled with Cycle-Detecting Depth Increment Algorithm.

Theorem 5.4.12. *The* CM Algorithm *coupled with* Cycle-Detecting Depth Increment Algorithm *satisfies the reachability property.*

Proof. Consider two trees, T_1 and T_2, both (connected) subgraphs of an underlying graph G. For each robot, i, let $R^{[i]}_{\text{pref}}$ be set in the following way. Let ${p_1}^{[i]} = f^{[i]}_{\text{p}}$ under T_1, ${p_2}^{[i]} = f^{[i]}_{\text{p}}$ under T_2. Let $({p_2}^{[i]}, {p_1}^{[i]}) \in R^{[i]}_{\text{pref}}$ and $({p_1}^{[i]}, j) \in R^{[i]}_{\text{pref}}$ for all $j \notin \{{p_2}^{[i]}, {p_1}^{[i]}\}$. No node will ever attach to a parent other than its parent in one of $\{T_1, T_2\}$ (since it starts attached to its T_1 parent, only prefers its T_2 parent over that, and the graph doesn't change). At all times, T_2 consists of pairs of the forms (i, j) where either

(i) $j = f^{[i]}_{\text{p}}$,

(ii) or j is a descendant of i in the current tree, or

(iii) j is not a descendant of i in the current tree.

To show that T_2 cannot consist solely of cases (i) and(iii) unless T_2 is the current tree we use Lemma 5.4.11, which shows that if there are any edges in case (iii), and no edges in case (ii), then the result is a disconnected graph with at least one cycle. So whenever there is some $(i, j) \in \mathcal{E}(T_2)$ where (i, j) is not in the current tree, there must be at least one such (i, j) where j is not a descendant of i. By Theorem 5.4.10 one such edge will be added to the current tree in finite time. This will stop happening when each $(i, f_{\mathrm{p}}^{[i]})$ in the current constraint tree is also in T_2. □

5.5 Simulations

Here we illustrate the performance of the Connectivity Maintenance Algorithm in several simulations. We combine the algorithm with Cycle-Detecting Depth Increment Algorithm and the deployment algorithm presented in [12]. The proximity graph of the robotic network is the r-disk proximity graph. The deployment algorithm assumes that each robot has a sensor coverage disk – i.e. the sensor we are interested in on each robot covers a disk of radius r centered about the robot position. The algorithm moves the robots to maximize sensor coverage of a "region of interest" represented by a density function $\rho : \mathbb{R}^2 \to \mathbb{R}$. In particular, it maximizes the integral, over the union of all sensor coverage disks, of the density function. We assume the robots have a maximum velocity of $v_{\max}$.

Definition 5.5.1. *Our deployment algorithm consists of the sets:*

(i) $L = \mathbb{Z}_n \times \mathbb{R}^2$*;*

(ii) $W = \mathbb{R}^2$*,* $w^{[i]}$ *contains the "target" of* i*'s motion,* $P_{targ}i$*;*

(iii) $W_0^{[i]} = W$*,* $i \in I$*;*

(iv) $W_{in} = \mathbb{Z}_n \times \mathbb{F}(\mathbb{Z}_n) \times \mathbb{F}(\mathbb{Z}_n)$, ${w_{in}}^{[i]} = (f_p^{[i]}, S^{[i]}_{constraints}, S^{[i]}_{children})$ *for* $i \in I$.

(v) $W_{out} = R(\mathbb{Z}_n, \mathbb{Z}_n)$, ${w_{out}}^{[i]} = (R^{[i]}_{pref})$ *defined as follows. For agent* $i \in \mathbb{Z}_n$,

$$(j,k) \in R^{[i]}_{pref} \text{ for } j,k \neq i \text{ if } n^{[i]}_{root} = n^{[j]}_{root} = n^{[k]}_{root} \text{ and } \|x^{[j]} - x^{[i]}\| < \|x^{[k]} - x^{[i]}\|.$$

(vi) ${W_{in,0}}^{[i]} \subseteq W_{in}$, *subsets of* allowable initial input values *Initially* $S^{[i]}_{constraints} = \mathcal{N}(i), f_p^{[i]} = i, S^{[i]}_{children} = \emptyset$;

and of the maps:

(i) $\text{msg}(x^{[i]}, w^{[i]}, {w_{in}}^{[i]}, i) = (i, x^{[i]})$;

(ii) $\text{stf}_{\text{io}}(w^{[i]}, {w_{in}}^{[i]},) =$

Let $D = \{p \in \mathbb{R}^2 \ : \ \|p - x^{[i]}\| \leq r_d \wedge \|p - x^{[i]}\| \leq \|p - x^{[j]}\| \forall j \in \mathcal{N}(i)\}$

Set $P_{targ}i \leftarrow [\frac{\int\int_D x\rho(x,y)dxdy}{\int\int_D \rho(x,y)dxdy}, \frac{\int\int_D y\rho(x,y)dxdy}{\int\int_D \rho(x,y)dxdy}]^T$

Let $D_C = \cap_{j \in \{f_p^{[i]}\} \cap S^{[i]}_{constraints} \cap S^{[i]}_{children}} B(\frac{x^{[j]}+x^{[i]}}{2}, \frac{r}{2})$ {Recall we are working with the r-disk graph}

Set $P_{targ}i \leftarrow \text{argmin}_{p \in D_C}(\|p - P_{targ}i\|)$

(iii) $\text{ctl}(x^{[i]}, x^{[i]}(t_{last}), w^{[i]}, {w_{in}}^{[i]}) = v_{max}(\frac{P_{targ}i - x^{[i]}}{\|P_{targ}i - x^{[i]}\|})$

Links of the constraint tree are preserved adapting the procedure described in [2]. To preserve a link between two robots, we constrain the motion of the two robots to a circle of radius $\frac{r}{2}$ centered at the midpoint of the line between their positions. Because each robot has a "target" it moves towards, we can find the closest point to the target in the intersection of the circles generated by the constraint edges. We do this by considering as candidate points each intersection between the boundaries of each pair of constraint circles, and each closest point between a given constraint circle and the target as shown in Figure 5.2. By filtering out those candidate points which are outside one or more constraint circles,

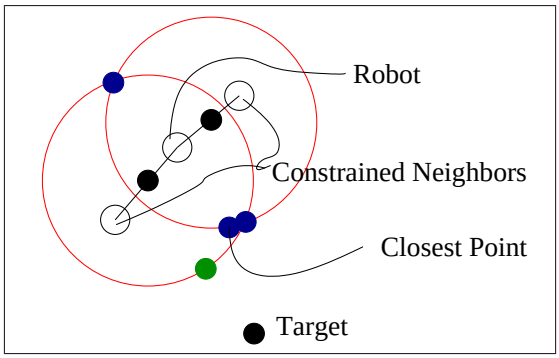
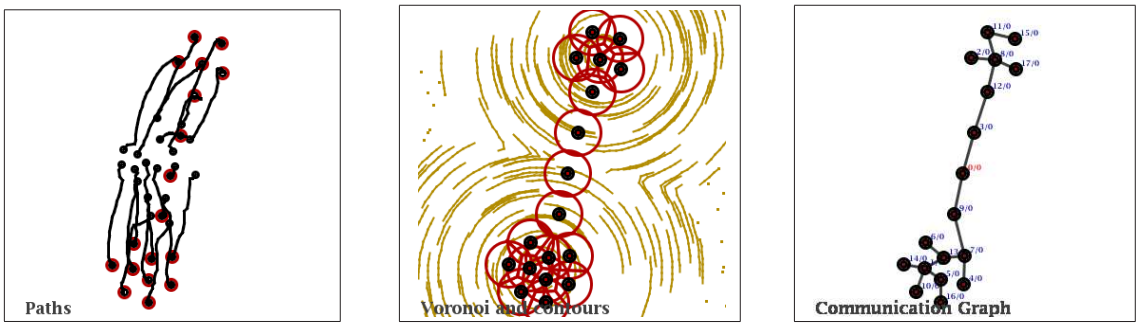


Figure 5.2: Motion constraints used in simulation.

and picking the remaining point with the minimum distance to the target, we find the closest feasible point to the target. Each robot is then instructed to move towards its closest feasible point instead of its original target. The constraint algorithm is made more robust by adding an extra constraint circle, centered about the position of the robot making the motion decision, with a radius equal to its maximum travel range between communication rounds.

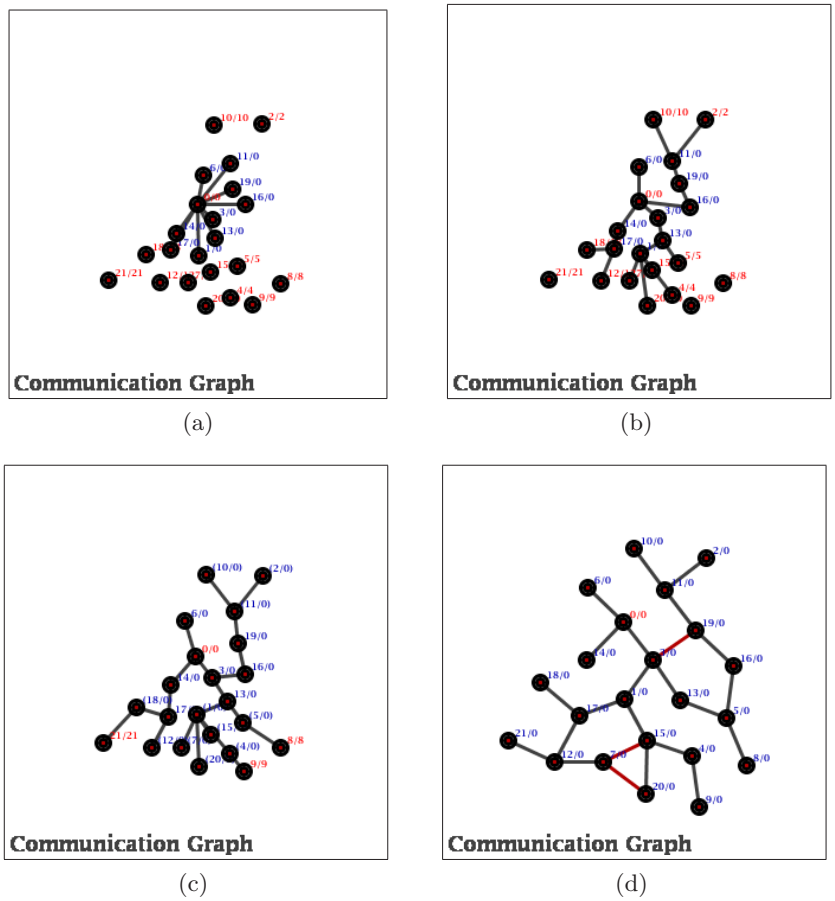
The algorithm resulting from the combination of the deployment algorithm with the Connectivity Maintenance Algorithm is executed in our Java simulation platform [53] for robotic networks. This platform has been developed to provide a software implementation of the modeling framework introduced in [37]. Our results are shown in Figure 5.3.



(a) (b) (c)

Figure 5.3: The plots show an execution of Connectivity Maintenance Algorithm, showing (a) the paths taken by the robots, (b) a contour plot of the density field and the sensor coverage regions of the robots, (c) the final network constraint tree.

The evolution of CM Algorithm coupled with Cycle-Detecting Depth Increment Algorithm and our deploy algorithm when repairing an initially disconnected tree is shown in Figure 5.4.



(a) (b)

(c) (d)

Figure 5.4: From left to right, top to bottom, progress of repair starting with an initially disconnected constraint tree. Agents are labeled by "agent id/root id" : those in red have not yet completed repair.

5.6 Conclusions and future work

We have studied connectivity maintenance of robotic networks performing spatially distributed tasks. We have proposed a distributed strategy based on maintaining the links of an evolving tree that is specified by the motion control algorithm. We have analyzed the correctness of the proposed algorithm and characterizes its repair and reachability and properties. All the developments have been done in a general modeling framework that allos for easy composition with other coordination algorithms.

5.7 Formation morphing problem

In this section, we illustrate the utility of the CONNECTIVITY MAINTENANCE introduced in Section 5.1.1. We design a coordination algorithm that provides the network with the capability of moving between any two different formations while maintaining connectivity. Let us introduce some useful task notions.

Definition 5.7.1 (Formation morphing problem)**.** *Given a proximity graph, $\mathcal{G}$, the formation morphing problem is that of designing a distributed algorithm to compute motion between two configurations, $[(P_s, T_s)]$ and $[(P_{targ}, T_{targ})]$, of n robots in d-dimensional space such that the graph $\mathcal{G}(i_{\mathbb{F}}(P))$ remains connected at all times and the network reaches $[(P_{targ}, T_{targ})]$ in finite time.*

5.7.1 Algorithm framework and specification

In this section, we introduce the FORMATION MORPHING ALGORITHM to solve the formation morphing problem.

For clarity, we will use the symbols $a_{i,1}, \ldots, a_{i,m_i}$, where m_i is the depth of i in the target constraint tree, T_{targ}, to denote the ancestors of i in T_{targ}. We use $a_{i,1}$ to mean the parent of i in T_{targ}, and a_{i,m_i} to mean the root of T_{targ}.

Consider a uniform network $\mathcal{S}$ with identifiers $I = \{0, \ldots, n-1\}$, agents of the form $A = (\mathbb{R}^2, \mathbb{R}^2, \mathbb{R}^2, f(x,u) = u)$, and r-disk communication edge map E_{cmm}, that is, i and j are connected if $\|x^{[i]} - x^{[j]}\| \leq r$. We use several constants to specify values known a priori and used by each agent's control law, message generation function and state transition function. Pick some lower numbers $d_1 < d_2 \leq r$ to be the distance constraints used in practice, and let all robots know d_1 and d_2 before commencing the morph algorithm. Each robot, i, also knows its position $(x_{\text{final}}, y_{\text{final}})$ in the final configuration;

A *formation morphing input-output control and communication law over* $\mathcal{S}$ consists of

(i) communication schedule equal to $\frac{1}{4}nt_{\text{cmm}}$ for $n \in \mathbb{N}$. This means that t_{cmm} is the time for CONNECTIVITY MAINTENANCE to go through all four iterations in a cycle. we also choose v_{scale} such that $4v_{\text{scale}}t_{\text{cmm}} < |d_2 - d_1|$;

(ii) communication language equal to

$$(i, \mathbb{I}_{\text{branch}}^{[i]}, n_{\text{chld}}^{[i]}, anc^{[i]}, x^{[i]}) \in$$

$$\mathbb{N} \times \{\text{true}, \text{false}\} \times \mathbb{Z}_n \times \mathbb{N}^3 \times \mathbb{R}^2,$$

where $(i, \mathbb{I}_{\text{branch}}^{[i]})$ is the logic variable state of the sending robot, i, $n_{\text{chld}}^{[i]}$ is the number of children i will have in the target configuration, and anc consists of 3 numbers used to indicate ancestry information (see Theorem 5.7.2);

(iii) logic variables, $W^{[i]} = \mathbb{Z}_n \times \{\text{true}, \text{false}\} \times \mathbb{Z}_n \times \mathbb{N}^3 \times \mathbb{Z}_3$, $w^{[i]} = (i, \mathbb{I}_{\text{branch}}^{[i]}, n_{\text{chld}}^{[i]}, anc^{[i]}, M_{\text{mode}}^{[i]})$, where i is the agent unique identifier, $\mathbb{I}_{\text{branch}}^{[i]}$ is a flag used to indicate which stage of the algorithm i is executing and $n_{\text{chld}}^{[i]}$ is the number of children of i in the target tree. Each robot, i, will know enough about its ancestors in the constraint tree, T_{targ}, of the target configuration, $[(P_{\text{targ}}, T_{\text{targ}})]$ to answer membership queries of the form "is j an ancestor of i?" and distance queries of the form "given that j is an ancestor of i, how many edges are in the shortest path from i to j in T_{targ}?" We show in Theorem 5.7.2 that this information can be stored with three integers which we will denote by $anc^{[i]}$. $M_{\text{mode}}^{[i]} \in \mathbb{Z}_3$ is used to indicate to the control function which of 3 motion modes to use. We will denote these modes M_{stay}, M_{origin} and $M_{\text{u,v final}}$, corresponding to a stationary mode, a "move towards origin" mode and a "move towards final configuration" mode;

(iv) ${W_0}^{[i]}$ equal to $\{(i, \text{false}, n_{\text{chld}}^{[i]}, anc^{[i]}, M_{\text{stay}})\}$;

(v) ${W_{\text{in}}}^{[i]}$ equal to $\mathbb{Z}_n$, where ${w_{\text{in}}}^{[i]} = f_{\text{p}}^{[i]}$, indicates the ith robots parent in the constraint tree;

(vi) ${W_{\text{in0}}}^{[i]}$ such that the tree induced by $f_{\text{p}}^{[i]}$ is a connected spanning tree of the robotic network;

(vii) ${W_{\text{out}}}^{[i]} = \{{w_{\text{out}}}^{[i]} : \mathbb{Z}_n \mapsto \mathbb{Z}_n \cup \{\infty\}\}$, where ${w_{\text{out}}}^{[i]} \in {W_{\text{out}}}^{[i]}$ is a ranking of which nearby robots i would prefer to be connected to;

and with the following functions

(i) the standard message generation function (i.e., $\text{msg}(t, x^{[i]}, w^{[i]}, j) = (x^{[i]}, w^{[i]})$);

(ii) **function** $\text{stf}_{\text{io-out}}(t, x^{[\text{id}]}, f_{\text{p}}^{[\text{id}]}, (\text{id}, \mathbb{I}_{\text{branch}}^{[\text{id}]}, anc^{[\text{id}]}, M_{\text{mode}}^{[\text{id}]}),$
$\{l_j \mid j \in \mathcal{N}_{\text{id}}\})$

1: Let ${w_{\text{out}}}^{[\text{id}]} : \mathbb{Z}_n \mapsto \mathbb{Z}_n \cup \{\infty\}$ be defined by ${w_{\text{out}}}^{[\text{id}]}(j)$

$$\begin{cases} \infty & \|x^{[j]} - x^{[\text{id}]}\| > d_1 \\ k & j = a_{\text{id},k} \wedge (\mathbb{I}_{\text{branch}}^{[i]} = \mathbb{I}_{\text{branch}}^{[j]} \vee j = f_{\text{p}}^{[\text{id}]}) \\ m_{\text{id}} + 1 & j \notin \{a_{\text{id},k} \ : \ k \in 1 \dots m_{\text{id}}\} \wedge j = f_{\text{p}}^{[f_{\text{p}}^{[\text{id}]}]} \\ m_{\text{id}} + 2 & j \notin \{a_{\text{id},k} \ : \ k \in 1 \dots m_{\text{id}}\} \wedge j = f_{\text{p}}^{[\text{id}]} \\ \infty & \text{otherwise} \end{cases}$$

2: return ${w_{\text{out}}}^{[\text{id}]}$

In other words, if no member of $\{a_{\text{id},k} \ : \ k \in \{1, \dots, m_{\text{id}}\}\}$ is available, id would prefer to attach to $f_{\text{p}}^{[f_{\text{p}}^{[\text{id}]}]}$, and if $f_{\text{p}}^{[f_{\text{p}}^{[\text{id}]}]}$ is also not in reach, id will remain attached to $f_{\text{p}}^{[\text{id}]}$;

(iii) **function** $\text{stf}_{\text{io}-\text{slf}}(t, f_{\text{p}}^{[\text{id}]}, {w_{\text{in}}}^{[\text{id}]}, w^{[\text{id}]}, \{l_j \ : \ j \in \mathcal{N}_{\text{id}}\})$

1: **if** $\|x^{[\text{id}]} - x^{[f_{\text{p}}^{[\text{id}]}]}\| \geq d_1$ **then**

2: Set $M_{\text{mode}}^{[\text{id}]} \leftarrow M_{\text{parent}}$

3: **else if** $\mathbb{I}^{[\mathrm{id}]}_{\mathrm{branch}} \neq \mathrm{true}$ **then**

4: **if** $f^{[\mathrm{id}]}_{\mathrm{p}} = a_{\mathrm{id},1}$ **then**

5: **if** each j such that $f^{[j]}_{\mathrm{p}} = \mathrm{id}$ has sent $\mathbb{I}^{[j]}_{\mathrm{branch}} = \mathrm{true}$ and there are $n^{[\mathrm{id}]}_{\mathrm{chld}}$ such j **then**

6: Set $\mathbb{I}^{[\mathrm{id}]}_{\mathrm{branch}} \leftarrow \mathrm{true}$.

7: **end if**

8: **end if**

9: $M^{[\mathrm{id}]}_{\mathrm{mode}} \leftarrow M_{\mathrm{origin}}$

10: **else if** $\mathbb{I}^{[\mathrm{id}]}_{\mathrm{branch}} = \mathrm{true}$ **then**

11: $M^{[\mathrm{id}]}_{\mathrm{mode}} \leftarrow M_{\mathrm{u,v\ final}}$

12: **end if**

13: return $(\mathrm{id}, \mathbb{I}^{[\mathrm{id}]}_{\mathrm{branch}}, n^{[\mathrm{id}]}_{\mathrm{chld}}, anc^{[\mathrm{id}]}, M^{[\mathrm{id}]}_{\mathrm{mode}})$

(iv) the control function is $\mathrm{ctl}(x^{[\mathrm{id}]}, w^{[\mathrm{id}]}) = 0$ if $M^{[\mathrm{id}]}_{\mathrm{mode}} = M_{\mathrm{stay}}$, $\mathrm{ctl}(x^{[\mathrm{id}]}, w^{[\mathrm{id}]}) = v_{\mathrm{scale}}\,\mathrm{vers}(-x^{[\mathrm{id}]})$ if $M^{[\mathrm{id}]}_{\mathrm{mode}} = M_{\mathrm{origin}}$, and $\mathrm{ctl}(x^{[\mathrm{id}]}, w^{[\mathrm{id}]}) = v_{\mathrm{scale}}\,\mathrm{vers}(x_{\mathrm{final}} - x^{[\mathrm{id}]})$ if $M^{[\mathrm{id}]}_{\mathrm{mode}} = M_{\mathrm{u,v\ final}}$.

Next, we specify how the numbers $anc^{[i]} \in \mathbb{N}^3$ are initialized.

Prior to running any control algorithms, perform the following operations on the constraint tree, T_{targ}, of $[(P_{\mathrm{targ}}, T_{\mathrm{targ}})]$. Perform a depth-first search on T_{targ}. Mark each node, i, with the number of nodes visited before node i ($n_{\mathrm{visit}}(i)$), and the number of descendants of i in the tree T_{targ} ($n_{\mathrm{desc}}(i)$). Note that $n_{\mathrm{visit}}(i) + n_{\mathrm{desc}}(i)$ is the number of nodes visited before the first node after i that is not an ancestor of i is visited. Recalling that m_i is the depth in the final target tree of node i, let $anc^{[i]} \leftarrow (n_{\mathrm{visit}}(i), n_{\mathrm{desc}}(i), m_i)$.

Theorem 5.7.2. *The numbers $anc^{[i]} \in \mathbb{N}^3$, $i \in I$, allows* Formation Morphing Algorithm *to answer queries of the form "Is robot j $a_{i,d}$ in $[(P_{targ}, T_{targ})]$?" using only $O(\log(n))$ bits of storage in $O(1)$ time.*

Note that storing a unique identifier for each robot requires $O(\log(n))$ bits. The Formation Morphing Algorithm is the composition (in the sense of Definition 2.2.3) of Connectivity

MAINTENANCE with the formation morphing input-output law defined above.

5.7.2 Correctness analysis

We will now establish the correctness of FORMATION MORPHING ALGORITHM. Lemma 5.7.3 shows that we do not break any edges in the constraint tree. Lemma 5.7.4, Lemma 5.7.4, Lemma 5.7.6 and Lemma 5.7.7 establish that FORMATION MORPHING ALGORITHM performs the necessary topology re-arrangements for formation morphing. These topology lemmas and Lemma 5.7.8 lead up to Theorem 5.7.9 which establishes the correctness and time complexity of FORMATION MORPHING ALGORITHM.

Lemma 5.7.3. *While following* FORMATION MORPHING ALGORITHM *composed with* CONNECTIVITY MAINTENANCE*, no two robots that are connected in the constraint tree are ever d_2 apart.*

Proof. Let robot i be the parent of robot j in the constraint tree. Let t_0 be the first instant of time at which this edge is contained in the tree. Consider the distance $d_{ij}(t) = \|x^{[i]}(t) - x^{[j]}(t)\|$. If this edge is the result of a re-attach event, then d_{ij} must have been less than d_1 one round before the attach (by line `18:` of CONNECTIVITY MAINTENANCE, $\text{stf}_{\text{io-out}}$ of FORMATION MORPHING ALGORITHM and the fact that the connect step is one round later at line `6:` of CONNECTIVITY MAINTENANCE). By the definition of v_{scale}, j and i cannot have moved more then $\|d_2 - d_1\|$ further apart in the intervening round. If t_0 is the initial creation of the spanning tree, then we still have $\|x^{[i]}(t_0) - x^{[j]}(t_0)\| < d_1$. At every round of communication one of two things happens. If $d_{ij}(t) \leq d_1$, the robots cannot move more than $2v_{\text{scale}}t_{\text{cmm}}$ further apart before the next communication round. Since $v_{\text{scale}} \leq \frac{d_2 - d_1}{2t_{\text{cmm}}}$, d_{ij} will be less than d_2 by the next communication round. On the other hand, if $d_{ij}(t) > d_1$, robot j moves towards $x^{[i]}(t)$ until the next communication round with velocity v_{scale}. Since i is moving with velocity at most v_{scale}, d_{ij} can be at most $2v_{\text{scale}}t_{\text{cmm}} - d_1$ away by the next communication round. Since $v_{\text{scale}} \leq \frac{d_1}{2t_{\text{cmm}}}$,

d_{ij} will be smaller than d_2 at next communication round. □

Lemma 5.7.4. *In the execution of* Formation Morphing Algorithm *for each robot, i, for each descendant, j, of i, $\mathbb{I}^{[i]}_{branch} = true$ at time t_1 implies $\mathbb{I}^{[j]}_{branch} = true$ for all $t \geq t_1$.*

Proof. If any descendant, j, does not satisfy $f_{\mathrm{p}}^{[j]} = a_{j,1}$, then $\mathbb{I}^{[j]}_{\mathrm{branch}} \neq$ true. Since j cannot attach to any node k having $\mathbb{I}^{[k]}_{\mathrm{branch}} \neq \mathbb{I}^{[j]}_{\mathrm{branch}}$, and $\mathbb{I}^{[f_{\mathrm{p}}^{[j]}]}_{\mathrm{branch}}$ cannot be set to true until $\mathbb{I}^{[j]}_{\mathrm{branch}} =$ true, no robot, k, in the path from j to the root can have $\mathbb{I}^{[k]}_{\mathrm{branch}} =$ true and neither can i. □

Lemma 5.7.5. *In the execution of* Formation Morphing Algorithm *no robot, i, satisfies $\mathbb{I}^{[i]}_{branch} =$ true until each of its descendants, j, in T_{targ} satisfies $f_p^{[j]} = a_{j,1}$.*

Proof. Let t be the first time at which $\mathbb{I}^{[i]}_{\mathrm{branch}} =$ true. By Lemma 5.7.4, each robot j which is a descendant of i in the constraint tree at time t satisfies $f_{\mathrm{p}}^{[j]} = a_{j,1}$. Each such robot must also be a descendant of i in the target tree, otherwise the path from some j to i would contain a link from k to $f_{\mathrm{p}}^{[k]}$ where $f_{\mathrm{p}}^{[k]} \neq a_{k,1}$. Since each descendant, j, of i checks that it has $n^{[j]}_{\mathrm{chld}}$ children before setting $\mathbb{I}^{[j]}_{\mathrm{branch}} =$ true, and no j having $f_{\mathrm{p}}^{[j]} = a_{j,1}$ switches parents, the number of descendants of i in the constraint tree at time t is equal to the number of descendants in the target tree. □

Lemma 5.7.6. *Let $diam(P(t_0))$ be the initial diameter of the convex hull of the robot positions. Within $O(\frac{diam(P(t_0))}{t_{cmm} v_{scale}})$ rounds, each robot i is within d_1 of each robot in the current path from i to $a_{i,1}$, or satisfies $f_p^{[i]} = a_{i,1}$.*

Proof. Any robot i with $\mathbb{I}^{[i]}_{\mathrm{branch}} \neq$ true moves towards $(0, 0)$. $O(\frac{\mathrm{diam}(P(t_0))}{t_{\mathrm{cmm}} v_{\mathrm{scale}}})$ is the number of rounds required for the robots to rendezvous under this behavior. Clearly the condition $\|x^{[i]} - k\| < d_1$ holds for all i, and k having $\mathbb{I}^{[i]}_{\mathrm{branch}} = \mathbb{I}^{[k]}_{\mathrm{branch}} =$ false before rendezvous occurs. Each robot, k, along the path from i to the root satisfies $\mathbb{I}^{[k]}_{\mathrm{branch}} =$ false as $f_{\mathrm{p}}^{[i]} \neq a_{i,1}$. Each robot k along the path from the root to $a_{i,1}$ satisfies $\mathbb{I}^{[k]}_{\mathrm{branch}} =$ false for the same reason. □

Lemma 5.7.7. *Within* $4(K)$ *rounds of the condition from Lemma 5.7.6 holding, each robot,* i*, is at depth and depth estimate of at least* $\min(dpth^{[i]}_{T_{targ}}, \max(K - 2dpth^{[i]}_{T_{targ}}, 1))$.

Proof. We will proceed by induction on K and $\text{dpth}^{[i]}_{T_{\text{targ}}}$. As a base case, when $K = 0$, every non-root robot is at depth at least 1. Assume true for $K - 1$, thus after $4(K - 1)$ rounds, for each of i's final ancestors, $a_{i,k}$ (where k is the final distance from $a_{i,k}$ to i) was at depth at least $\min(\text{dpth}^{[a_{i,k}]}_{T_{\text{targ}}}, \max(K - 1 - \text{dpth}^{[a_{i,k}]}_{T_{\text{targ}}}, 1)) = \min(\text{dpth}^{[a_{i,k}]}_{T_{\text{targ}}}, \max(K - \text{dpth}^{[i]}_{T_{\text{targ}}} + (2k - 1), 1))$. If $\max(K - \text{dpth}^{[i]}_{T_{\text{targ}}}, 1) = 1$ or if i is at its final depth, there is nothing left to do, otherwise each of i's ancestors is at a depth greater then i, or at its final position. This means that the one ancestor of i at i's current depth and depth estimate is in its final position, and line 2 of stf_{io} of Connectivity Maintenance allows i to make the connection, increase its depth by 1 and over the next 4 rounds, increase its depth estimate to the proper value. □

Lemma 5.7.8. *Within* $dpth^{[i]}_{T_{targ}}(1 + \frac{2d_1}{t_{cmm}v_{scale}})$ *rounds of the first time* $\mathbb{I}^{[j]}_{branch}$ *is true for all* j*, where* T_{targ} *is the topology of the final configuration, each robot* i *is at its final position.*

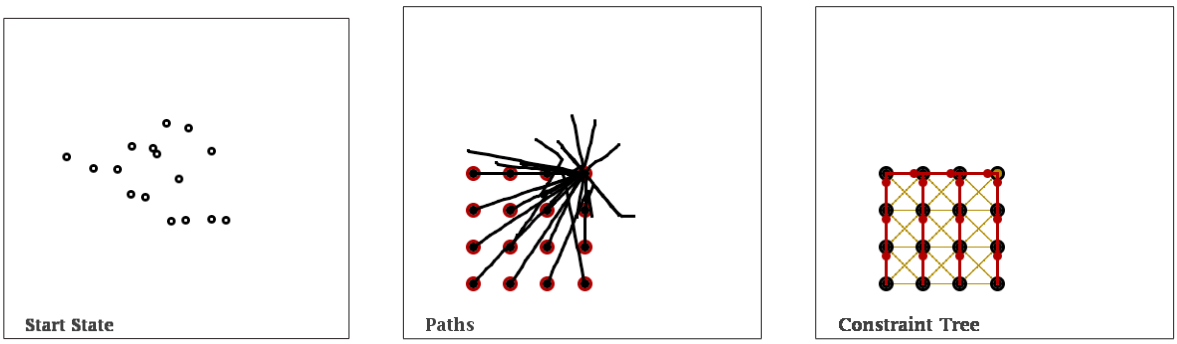
Proof. Let us induct on $\text{dpth}^{[i]}_{T_{\text{targ}}}$. As a base case, the root reaches its final position in zero rounds. When $f_{\text{p}}^{[i]}$ reaches its final position, it stops, and within one more round, i is within d_1 of $x^{[f_{\text{p}}^{[i]}]}$ which is within a distance of d_1 of i's final position. It takes i at most $\frac{2d_1}{t_{\text{cmm}}v_{\text{scale}}}$ further rounds to reach its final position. □

The next result follows from the previous discussion.

Theorem 5.7.9. *Within* $O(\frac{(diam(P(t_0)) + diam(T_{targ})d_1}{t_{cmm}v_{scale}})$ *rounds* Formation Morphing Algorithm *achieves formation morphing, where* T_{targ} *is the final tree in the target configuration, and* $diam(T_{targ})$ *is its graph diameter.*

5.8 Simulation results

We have developed a custom java simulation engine for robotic networks expressed in the formalism of [37]. We used this framework to develop simulations and visualizations of the Formation Morphing Algorithm. The source is available upon request.



(a) (b) (c)

Figure 5.5: Plots show (a) the initial positions, (b) the paths taken by and (c) the final configuration (including constraint tree) of an execution of Formation Morphing Algorithm.

We further developed the simulator to run approximately 80000 runs of Formation Morphing Algorithm with initial and final configurations sampled randomly. In Figure 5.6 we plot the actual task completion times of each of these runs versus the function $\text{diam}(P(t_0)) + \text{diam}(T_{\text{targ}})$. Because of the uniform time schedule, the number of communication rounds required for completion is linearly related to the time required for completion. From the graph one can see a linear relationship between the worst completion times for Formation Morphing Algorithm and $\text{diam}(P(t_0)) + \text{diam}(T_{\text{targ}})$, as fore-casted by our analysis.

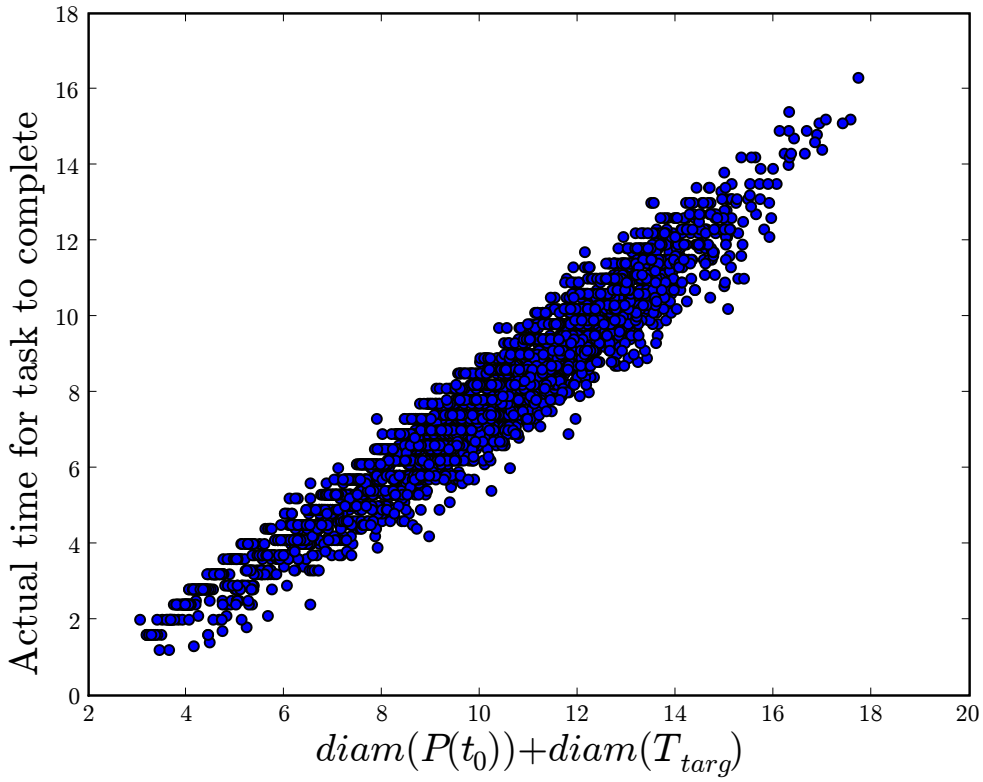


Figure 5.6: Comparison of the time complexity bound in Theorem 5.7.9 with actual running times of Formation Morphing Algorithm under random choices of initial and final configurations. Each point represents a successful execution.

Chapter 6

Algebraic connectivity maximization

6.1 Robotic network model and problem formulation

Assume we are given a specific algorithm which achieves a coordination task. Our objective is to design a procedure that modifies the directions of motion specified by the given algorithm as little as possible while preserving network connectivity. Let us start by formalizing this idea.

Definition 6.1.1. *An* Underlying Control Law *for n robots in $\mathbb{R}^d$ is a specification, for each network configuration $\mathcal{P}$, of a control input u_{goal-i} for each robot $i \in \{1, \ldots, n\}$, a bound θ_{max-i} on the angle by which the true motion of the ith robot is allowed to deviate from u_{goal-i}, and a time step $\delta T > 0$ over which u_{goal-i} and θ_{max-i} are valid. A set of inputs $(u_i)_{i\in\{1,\ldots,n\}}$ is* compatible with *the* Underlying Control Law *if and only if the following two conditions hold for all $i \in \{1, \ldots, n\}$,*

$$\|u_i\| \leq \|u_{goal-i}\|, \qquad |\angle(u_{goal-i}, u_i)| \leq \theta_{max-i}. \qquad \bullet$$

The first problem we address is that of deciding when a proposed motion can be made while safely maintaining connectivity of the robotic network.

Problem 6.1.2 (Spectral Connectivity Decision Problem)**.** *Given a control input, u_i known to the ith agent, for $i \in \{1, \ldots, n\}$, a control bound, v_{max}, known to all agents, such that each agent's control input, u_i must always satisfy $\|u_i\| \leq v_{max}$, a time interval $[t_0, t_0 + \delta T]$, and $[\lambda_-, \lambda_+] \subset \mathbb{R}_{>0}$,* Spectral Connectivity Decision Problem *consists of providing a (distributed) procedure which, for each robot i returns a value, $f_{safe} \in \mathbb{R}$ having $f_{safe} \geq 0$ only if the following hold for all $t \in [t_0, t_0 + \delta T]$ for all $\{\tilde{u}_j, \|\tilde{u}_j\| \leq v_{max}\}_{j \in \{1,\ldots,n\} \setminus \{i\}}$ and for all network configurations, $\mathcal{P}$, consistent with the information available to the ith robot.*

- $f_{2-conn}(\mathcal{P}(t)) \notin [\lambda_-, \lambda_+]$, *or*
- $f^{\circ}_{2-conn}(\mathcal{P}(t); [\mathbf{0}, \ldots, u_i^T, \ldots, \mathbf{0}]^T) \geq 0$. •

where $\mathcal{P}(t)$ denotes the network evolution under control $(\tilde{u}_1, \ldots, \tilde{u}_{i-1}, u_i, \tilde{u}_{i+1}, \ldots, \tilde{u}_n)$ starting from $\mathcal{P}$.

We follow with an informal description of Definition 6.1.2

> *[Informal description:]* Our goal is to determine (in a distributed manner) whether a given motion could, potentially, cause λ_2 to drop below the threshold λ_+. Under ideal conditions with exact computation, it would suffice to ensure that no motion causes the time derivative of λ_2 to be less than zero whenever λ_2 is exactly equal to λ_+. Because exact computation is, at best, infeasible (and at worst impossible), we instead ensure that the time derivative of λ_2 is never less than zero whenever λ_2 inhabits a range around λ_+, in this case $[\lambda_-, \lambda_+]$ (while we have not characterized an optimal value for λ_- given λ_+, in practice $\lambda_- = 0.0$ works just fine, and elegantly handles cases where the initial connectivity is lower than planned for). We further specify that our solution answer this decision problem by returning a number, f_{safe}, which is greater than or equal to zero only if we can guarantee that our proposed robot motion is safe.

The second problem we address is the problem of determining directions of motion that are compatible with the given algorithm and preserve connectivity.

Problem 6.1.3 (Spectral Connectivity Problem)**.** *Given an* Underlying Control Law*, a bound, v_{max}, on agent control input, a time interval, $[t_0, t_0 + \delta T]$, and an interval $[\lambda_-, \lambda_+] \subset \mathbb{R}_{>0}$,*

Spectral Connectivity Problem *consists of providing a procedure which, for each robot,* $i \in \{1, \ldots, n\}$ *finds an input* u_i*, having* $\|u_i\| \leq v_{max}$*, compatible with the* Underlying Control Law *such that* Spectral Connectivity Decision Problem *returns a value* $f_{safe} \geq 0$ *when provided with* $[t_0, t_0 + \delta T]$*,* $[\lambda_-, \lambda_+]$*, and* u_i*.* •

For clarity, we also present an informal description of Definition 6.1.3

[Informal description:] In Definition 6.1.3 we are describing the process of, given a procedure to solve Spectral Connectivity Decision Problem, determine whether we can find a set of robot motions close to the motions specified by the underlying control law which are allowed by our solution to Spectral Connectivity Decision Problem. In our case, "close" means "the direction taken by each robot is close in angle to the direction proposed by the underlying control law." This is somewhat like expressing Spectral Connectivity Decision Problem as a function from "angle of motion" to f_{safe}, and searching for roots of f_{safe}.

6.2 Eigenvalue games and information dissemination

In this section we introduce the main algorithmic components of our solution to the problems presented in Section 6.1. In Section 6.2.1, we reformulate Spectral Connectivity Decision Problem as a game, termed Graph Picking Game, which can be played with out-of-date information on the state of the network and in Section 6.2.2 we study the properties of its space of solutions. Next, we present in Section 6.2.3 a distributed procedure that allows network agents to decide whether an intended motion wins Graph Picking Game. The other algorithmic component of our solution is a distributed information dissemination algorithm, presented in Section 6.2.4, which provides each robot with the information needed to play Graph Picking Game.

6.2.1 Graph Picking Game

We are interested in characterizing the rates of change of Laplacian matrices arising from instantaneous robot motions which solve Spectral Connectivity Decision Problem. To do this, we reformulate this problem as a game and study the properties of the solutions to the game.

In order to present a clean formulation, let us introduce some notation. Let

$$\mathrm{LAP}_{\pm}(n) = \{M \in \mathrm{Sym}(n) \ : \ M\mathbf{1} = \mathbf{0}\},$$

$$\mathrm{LAP}(n) = \{M \in \mathrm{LAP}_{\pm}(n) \ : \ M_{i,j} \leq 0 \text{ for all } i \neq j\}.$$

Note that, given $M \in \mathrm{LAP}(n)$, it is possible to define a graph, G, with Laplacian M, by assigning to each edge $(i,j) \in \mathcal{E}$ the weight $-M_{i,j}$. We consider the following partial order in $\mathrm{LAP}_{\pm}(n)$. For $A, B \in \mathrm{LAP}_{\pm}(n)$, we write $A <_{\mathrm{LAP}} B$ if and only if $A_{i,j} > B_{i,j}$ for all $i \neq j \in \{1, \dots n\}$. Likewise, $A \leq_{\mathrm{LAP}} B$ if and only if $A_{i,j} \geq B_{i,j}$ for all $i \neq j \in \{1, \dots n\}$. For $A \leq_{\mathrm{LAP}} B$, we define the interval

$$[A, B]_{\mathrm{LAP}} = \{L \in \mathrm{LAP}_{\pm}(n) \ : \ A \leq_{\mathrm{LAP}} L \leq_{\mathrm{LAP}} B\}.$$

Note that $A, B \in \mathrm{LAP}(n)$ and $L \in [A, B]_{\mathrm{LAP}}$ imply $L \in \mathrm{LAP}(n)$. The following result provides more properties of the matrices in the interval $[A, B]_{\mathrm{LAP}}$.

Lemma 6.2.1. *Let $A, B \in LAP(n)$ and $L \in [A, B]_{LAP}$. Then,*

(i) $f_{\lambda_2}(L) \in [f_{\lambda_2}(A), f_{\lambda_2}(B)]$,

(ii) $vv^T \bullet L \in [vv^T \bullet A, vv^T \bullet B]$ *for* $v \in \mathbb{R}^n$.

Proof. Fact (i) follows from the monotonicity of $\lambda_2(G)$ on the edge weights of G. To prove fact (ii), note that $vv^T \bullet L = v^T L v$ for any $L \in \mathrm{Sym}(n)$ and any $v \in \mathbb{R}^n$. Because any graph Laplacian is positive

semidefinite, and $L - A, B - L \in \mathrm{LAP}(n)$, we have

$$vv^T \bullet (L - A) = vv^T \bullet L - vv^T \bullet A \geq 0,$$

$$vv^T \bullet (B - L) = vv^T \bullet B - vv^T \bullet L \geq 0,$$

and the result follows. □

We can now formulate the core question we need to solve to obtain a solution to Spectral Connectivity Decision Problem as a game played against a graph-picking opponent.

Definition 6.2.2 (Graph Picking Game)**.** *Given $A, B \in LAP(n)$ with $A \leq_{LAP} B$, we pick $Y \in [A, B]_{LAP}$. Our opponent then selects $L \in [A, B]_{LAP}$. We win if either of the following conditions hold*

- $f_{\lambda_2}(L) \notin [\lambda_-, \lambda_+]$, *or*
- $f^{\circ}_{\lambda_2}(L; Y) \geq 0$. •

Our objective is to characterize the choices Y that ensure that Graph Picking Game is won (specifically, we characterize the choices X such that all Y having $X \leq_{\mathrm{LAP}} Y$ win Graph Picking Game). This is what we tackle next.

6.2.2 Bounds on matrices which win Graph Picking Game

A direction that a robot can take in physical space induces an instantaneous rate of change of the Laplacian matrix of the underlying communication graph of the network. Given out-of-date information on the state of the network, each robot can produce bounds on the actual Laplacian of the graph. In this section we answer the following question: given a matrix lower bound $A \in \mathrm{LAP}(n)$ and a matrix upper bound $B \in \mathrm{LAP}(n)$ on the Laplacian matrix of the communication graph and a range of

possible instantaneous rates of change of the Laplacian matrix due to a proposed physical motion, can we guarantee that the proposed motion will not decrease the second smallest eigenvalue of the graph Laplacian? We do this by answering the related question: given the information listed above, and a range of "unsafe" eigenvalues, $[\lambda_-, \lambda_+]$, can we guarantee the proposed motion will not decrease the second smallest eigenvalue of the Laplacian matrix whenever the said eigenvalue is outside of the range $[\lambda_-, \lambda_+]$?

More formally, we bound the union of all possible gradients of f_{λ_2} evaluated at $L \in [A, B]_{\text{LAP}}$. Consider $L \in [A, B]_{\text{LAP}}$ such that $f_{\lambda_2}(L) \in [\lambda_-, \lambda_+]$. Following the formula for the gradient of f_{λ_2} in Theorem 2.6.3, we examine the vectors $w \in \mathbb{S}^n$ such that $Lw = \lambda_2(L)w$. For such vectors, we have $L \bullet (ww^T) = w^T L w = f_{\lambda_2}(L)$, and therefore, using Lemma 6.2.1, it follows that $A \bullet (ww^T) \leq \lambda_+$. Our strategy is then to bound the set of $w \in \mathbb{R}^n$ which satisfy $A \bullet (ww^T) \leq \lambda_+$. The fact that the nonsmooth gradient of f_{λ_2} is actually the convex closure of such ww^T is addressed in Proposition 6.2.3.

Let $\{u_1, \ldots, u_m\}$ be the m eigenvectors of A corresponding to eigenvalues $\lambda_j \leq \lambda_+$ and let $\{u_{m+1}, \ldots, u_n\}$ be the $n - m$ eigenvectors of A corresponding to eigenvalues $\lambda_j > \lambda_+$. Given $\tilde{m} \geq m$, define

$$\epsilon_A(\tilde{m}) = \sqrt{\frac{\lambda_+ - \lambda_2(A)}{\lambda_{\tilde{m}+1}(A) - \lambda_2(A)}},$$

$$\mathbf{u}_{\text{span-}A}(\tilde{m}) = \text{span}\{u_1, \ldots, u_{\tilde{m}}\},$$

$$U_A(\tilde{m}) = \{w \in \mathbb{S}^n \mid \exists u \in \mathbf{u}_{\text{span-}A}(\tilde{m}) \text{ with } \|u\| = 1 \text{ such that } w \in B(u, \epsilon_A(\tilde{m}))\}.$$

We have chosen $U_A(\tilde{m})$ to contain the elements w satisfying $ww^T \in \partial f_{\lambda_2}(L)$ for any $\tilde{m} \geq m$, as we show next.

Proposition 6.2.3. *Let $A, B \in LAP(n)$, $L \in [A, B]_{LAP}$, $f_{\lambda_2}(L) \leq \lambda_+$, $w \in \mathbb{S}^n$, $\tilde{m} \geq m$. If $w \notin U_A(\tilde{m})$,*

then $ww^T \notin \partial f_{\lambda_2}(L)$. •

Proof. If $w \notin U_A(\tilde{m})$, then the component of w outside $\text{span}(v_1(A), \ldots, v_{\tilde{m}-1}(A))$ has magnitude at least $\epsilon_A(\tilde{m})$. Since $w \in \mathbb{S}^n$, the remaining component has magnitude at most $\sqrt{1-\epsilon_A^2(\tilde{m})}$, and therefore we can deduce

$$\begin{aligned} w^T A w &> (\sqrt{1-\epsilon_A(\tilde{m})^2})^2 f_{\lambda_2}(A) + \epsilon_A(\tilde{m})^2 f_{\lambda_{\tilde{m}+1}}(A) \\ &= f_{\lambda_2}(A) + \epsilon_A(\tilde{m})^2 (f_{\lambda_{\tilde{m}+1}}(A) - f_{\lambda_2}(A)) = \lambda_+. \end{aligned}$$

Since $L \in [A,B]_{\text{LAP}}$, $w^T L w \geq w^T A w > \lambda_+$. By $\lambda_+ > f_{\lambda_2}(L)$, $w \notin \{v \in \mathbb{S}^n \ : \ v^T L v = f_{\lambda_2}(L)\}$. To show $ww^T \notin \partial f_{\lambda_2}(L)$ we recall that $\partial f_{\lambda_2}(L) = \text{co}_{\{v \in \mathbb{S}^n \,:\, Lv = f_{\lambda_2}(L)v\}}\{vv^T\} = \text{co}_{\{v \in \mathbb{S}^n \,:\, v^T L v = f_{\lambda_2}(L)\}}\{vv^T\}$. Noting that there is only one combination of vectors in $\{vv^T \ : \ v \in \mathbb{S}^n\}$ which can have a convex combination of ww^T for $w \in \mathbb{S}^n$, namely the singleton set $\{ww^T\}$, we deduce that $w \notin \{v \in \mathbb{S}^n \ : \ v^T L v = f_{\lambda_2}(L)\}$ implies $ww^T \notin \text{co}_{\{v \in \mathbb{S}^n \,:\, v^T L v = f_{\lambda_2}(L)\}}\{vv^T\} = \partial f_{\lambda_2}(L)$. □

The bound induced by $U_A(\tilde{m})$ works for any $\tilde{m} \geq m$. Our idea is to check the bounds induced by all such $\tilde{m} \geq m$, in the hope of finding one which verifies that our proposed motion is allowable.

The following result is a consequence of Proposition 6.2.3 and Theorems 2.6.3 and 2.6.4.

Corollary 6.2.4. *Any instantaneous change in robot positions,* $(u_i)_{i \in \{1,\ldots,n\}}$*, inducing an instantaneous rate of change of the Laplacian* $Y \in LAP_{\pm}(n)$ *satisfying* $Y \bullet (uu^T) \geq 0$ *for all* $u \in U_A(\tilde{m})$ *for some* $\tilde{m} \geq m$*, satisfies* $f^{\circ}_{2-conn}(\mathcal{P}; (u_i)_{i \in \{1,\ldots,n\}}) \geq 0$. •

Given some $\tilde{m} \geq m$, we can conclude from Proposition 6.2.3 and Corollary 6.2.4 that any Y satisfying $Y \bullet (ww^T) \geq 0$ for all $w \in U_A(\tilde{m})$ wins Graph Picking Game on $A, B, \lambda_-, \lambda_+$. To determine whether a given Y satisfies this property, it is sufficient to find the vector $u_{\min} \in \mathbf{u}_{\text{span-}A}(\tilde{m})$

which minimizes $Y \bullet (uu^T) = u^T Y u$ (and then tack a fudge factor based on $\epsilon_A(\tilde{m})$ onto this minimum). We justify this in the following.

Let $M_u(\tilde{m}) \in \mathbb{R}^{n \times \tilde{m}}$ be a matrix whose column vectors are an orthonormal basis of $\mathbf{u}_{\text{span-}A}(\tilde{m})$. Note that any vector in $\mathbf{u}_{\text{span-}A}(\tilde{m}) \cap \mathbb{S}^n$ can be expressed in the form $M_u(\tilde{m})x$ for some $x \in \mathbb{S}^{\tilde{m}}$. Likewise any $x \in \mathbb{S}^{\tilde{m}}$ satisfies $M_u(\tilde{m})x \in \mathbf{u}_{\text{span-}A}(\tilde{m}) \cap \mathbb{S}^n$.

Proposition 6.2.5. *Finding the vector $u \in \mathbf{u}_{span\text{-}A}(\tilde{m}) \cap \mathbb{S}^n$ which minimizes $Y \bullet (uu^T)$ is equivalent to finding the vector $x \in \mathbb{S}^{\tilde{m}}$ which minimizes $x^T M_u^T(\tilde{m}) Y M_u(\tilde{m}) x$. Since $M_u^T(\tilde{m}) Y M_u(\tilde{m})$ is symmetric, minimization is achieved when $u^T Y u$ equals the smallest eigenvalue of $M_u^T(\tilde{m}) Y M_u(\tilde{m})$.* •

Proof. Let $u_{\min}$ be the vector in $\mathbf{u}_{\text{span-}A}(\tilde{m}) \cap \mathbb{S}^n$ that minimizes $u^T Y u$. Since there exists $x_{\min} \in \mathbb{S}^{\tilde{m}}$ having $M_u(\tilde{m})x_{\min} = u_{\min}$, we have $x_{\min}^T M_u^T(\tilde{m}) Y M_u(\tilde{m}) x_{\min} \leq \min_{u \in \mathbf{u}_{\text{span-}A}(\tilde{m}) \cap \mathbb{S}^n}(u^T Y u)$, and hence $\min_{\lambda \in \text{eigs}(M_u^T(\tilde{m}) Y M_u(\tilde{m}))}(\lambda) \leq u_{\min}^T Y u_{\min}$. Since each $x \in \mathbb{S}^{\tilde{m}}$ satisfies $M_u(\tilde{m})x \in \mathbf{u}_{\text{span-}A}(m) \cap \mathbb{S}^n$, then $x^T M_u^T(\tilde{m}) Y M_u(\tilde{m}) x \geq u_{\min}^T Y u_{\min}$ for all $x \in \mathbb{S}^{\tilde{m}}$. Thus $\min_{\lambda \in \text{eigs}(M_u^T(\tilde{m}) Y M_u(\tilde{m}))}(\lambda) \geq u_{\min}^T Y u_{\min}$. We conclude that $\min_{\lambda \in \text{eigs}(M_u^T(\tilde{m}) Y M_u(\tilde{m}))}(\lambda) = u_{\min}^T Y u_{\min}$. □

The next results provides a sufficient criterion to check if a matrix is a winning solution to Graph Picking Game.

Proposition 6.2.6. $(1 - \epsilon_A(\tilde{m})^2) Y \bullet (uu^T) + \epsilon_A(\tilde{m})^2 \min(\min(\textit{eigs}(Y)), 0) \geq 0$ *for all $u \in \mathbf{u}_{span\text{-}A}(\tilde{m})$ implies that $Y \bullet (ww^T) \geq 0$ for all $w \in U_A(\tilde{m})$.*

Proof. Any $w \in U_A(\tilde{m})$ can be decomposed into $\alpha u + \sqrt{1 - \alpha^2} v$ for $u \in \mathbf{u}_{\text{span-}A}(\tilde{m})$ and $v \in \text{complement}(\mathbf{u}_{\text{span-}A}(\tilde{m}))$ where $\sqrt{1 - \alpha^2} \leq \epsilon_A(\tilde{m})$. Since Proposition 6.2.5 gives us $Y \bullet (vv^T) \geq \min(\text{eigs}(Y))$, we have $Y \bullet (\sqrt{1 - \alpha^2}^2 vv^T) \geq \epsilon_A^2(\tilde{m}) \min(\text{eigs}(Y))$ if $\min(\text{eigs}(Y)) \leq 0$ and $Y \bullet (\sqrt{1 - \alpha^2}^2 vv^T) \geq \min(\text{eigs}(Y)) \geq 0$ if $\min(\text{eigs}(Y)) \geq 0$. Thus $Y \bullet (\sqrt{1 - \alpha^2}^2 vv^T) \geq$

$\epsilon_A^2(\tilde{m})\min(\min(\text{eigs}(Y)),0)$ and $Y \bullet (ww^T) \geq (1-\epsilon_A(\tilde{m})^2)Y \bullet (uu^T) + \epsilon_A(\tilde{m})^2\min(\min(\text{eigs}(Y)),0) \geq 0$. □

6.2.3 Direction Checking Algorithm

We introduce Direction Checking Algorithm in Table 6.1. Given $A, B \in \text{LAP}(n)$, and a lower bound, $X \in \text{LAP}_{\pm}(n)$ of the candidate instantaneous rate of change of the Laplacian matrix, $Y \in \text{LAP}_{\pm}(n), Y \geq_{\text{LAP}} X$, the algorithm returns a value $S_{\text{check}} \geq 0$ if it can verify that any $Y \geq_{\text{LAP}} X$ wins Graph Picking Game on A and B, and returns $S_{\text{check}} < 0$ otherwise.

Lemma 6.2.7. Direction Checking Algorithm *returns $S_{check} \geq 0$ only if X satisfies $X \bullet M \geq 0$ for every $M \in \partial f_{\lambda_2}(L)$ with $L \in [A,B]_{LAP}$.* •

Proof. If Direction Checking Algorithm returns $S_{\text{check}} \geq 0$, Proposition 6.2.5 implies that there must be some $\tilde{m} \geq m$ having $\min_{\{ww^T \,:\, w \in B(u,\epsilon_A(\tilde{m})), u \in \mathbb{S}^n \cap \mathbf{u}_{\text{span-}A}(\tilde{m})\}}(X \bullet (uu^T)) \geq 0$ thus, by Proposition 6.2.6, $X \bullet ww^T \geq 0$ for all $w \in U_A(\tilde{m})$ and, by Proposition 6.2.3, $X \bullet ww^T \geq 0$ for all $w \in \mathbb{S}^n$ having $ww^T \in \partial f_{\lambda_2}(L)$. Since any $M \in \partial f_{\lambda_2}(L)$ is the convex combination of some set of $ww^T \in \partial f_{\lambda_2}(L)$, $X \bullet M$ must be the sum of $X \bullet (ww^T)$ for such a set of ww^T and thus $X \bullet M \geq 0$. □

The following result shows that Direction Checking Algorithm is successful in determining if we win Graph Picking Game.

Theorem 6.2.8. Direction Checking Algorithm *returns $S_{check} \geq 0$ only when each Y having $Y \geq_{LAP} X$ satisfies $Y \bullet M \geq 0$ for $M \in \partial f_{\lambda_2}(L)$ with $L \in [A,B]_{LAP}$.* •

Proof. The conditions $S_{\text{check}} \geq 0$ holds only if

$$\min(\text{eigs}(M_u^T(\tilde{m})XM_u(\tilde{m}))) + \epsilon_A(\tilde{m})\min(\lambda_{\min}(X),0) \geq 0,$$

Name: Direction Checking Algorithm

Goal: Let Y be the (unknown) instantaneous rate of change of the Laplacian matrix of the communication graph of a robotic network. Given X (known) such that $Y-X$ is known to be positive semidefinite, determine whether Y can be proved to win Graph Picking Game on A and B and eigenvalue bounds λ_- and λ_+

Inputs:
- Matrices $A, B \in \mathrm{LAP}(n)$
- Eigenvalue bounds $\lambda_- \leq \lambda_+ \in \mathbb{R}$
- Lower bound, $X \in \mathrm{LAP}_{\pm}(n)$, on candidate direction in matrix space, $Y \in \mathrm{LAP}_{\pm}(n)$

Outputs: $S_{\text{check}} \in \mathbb{R}$. $S_{\text{check}} \geq 0$ means each $Y \geq_{\text{LAP}} X$ wins Graph Picking Game on A, B and $[\lambda_-, \lambda_+]$

Let $\lambda_+ \leftarrow \min(\lambda_+, \lambda_2(B))$
Let $\lambda_- \leftarrow \max(\lambda_-, \lambda_2(A))$
if $\lambda_- > \lambda_+$ **then**
  return 0
end if
Let $\lambda_{\min} \leftarrow \min(\mathrm{eigs}(X))$
Let $m_{\min} \leftarrow \min\{m \ : \ \lambda_m \in \mathrm{eigs}(A), \ \lambda_m > \lambda_+\}$
Initialize $S_{\text{check}} \leftarrow -1$.
for all $\tilde{m} \in \{m_{\min} - 1, \ldots, n\}$ **do**
  if $\tilde{m} < n$ **then**
    Let $\epsilon_A(\tilde{m}) \leftarrow \sqrt{\frac{\lambda_+ - \lambda_2(A)}{\lambda_{\tilde{m}+1}(A) - \lambda_2(A)}}$ and $\mathbf{u}_{\text{span-}A}(\tilde{m}) \leftarrow \mathrm{span}(u_j, j \in \{1, \ldots, m\})$
  else
    Let $\epsilon_A(\tilde{m}) \leftarrow 0$
  end if
  Let $M_u(\tilde{m}) \in \mathbb{R}^{n \times \tilde{m}}$ whose columns are orthogonal basis of $\mathbf{u}_{\text{span-}A}(\tilde{m})$
  Let $S \leftarrow (1 - \epsilon_A(\tilde{m})^2) \min(\mathrm{eigs}(M_u^T(\tilde{m}) X M_u(\tilde{m}))) + \epsilon_A(\tilde{m})^2 \min(\lambda_{\min}, 0)$ {*Does current $\tilde{m}$ verify X is safe?*}
  Let $S_{\text{check}} \leftarrow \max(S, S_{\text{check}})$ {*Does any $\tilde{m}$ checked so far verify X is safe?*}
end for
return S_{check} {*Does any $\tilde{m}$ verify X is safe?*}

Table 6.1: Direction Checking Algorithm.

for some $\tilde{m} > m$. For any Y having $Y \geq_{\text{LAP}} X$, $Y - X$ is positive semidefinite, thus each eigenvalue of Y is greater than or equal to the corresponding eigenvector of X, making $\lambda_{\min}(Y) > \lambda_{\min}(X)$. Likewise, we can express $M_u^T(\tilde{m})YM_u(\tilde{m})$ as $M_u^T(\tilde{m})XM_u(\tilde{m}) + M_u^T(\tilde{m})(Y-X)M_u(\tilde{m})$ where $Y - X$, and therefore $M_u^T(\tilde{m})(Y-X)M_u(\tilde{m})$ as well, are each positive semidefinite. This means that each eigenvalue of $M_u^T(\tilde{m})YM_u(\tilde{m}) = M_u^T(\tilde{m})XM_u(\tilde{m}) + M_u^T(\tilde{m})(Y-X)M_u(\tilde{m})$ is greater than or equal to the corresponding eigenvalue of $M_u^T(\tilde{m})XM_u(\tilde{m})$. So $\min(\text{eigs}(M_u^T(\tilde{m})XM_u(\tilde{m}))) + \epsilon_A(\tilde{m})\min(\lambda_{\min}(X), 0)$ $\leq \min(\text{eigs}(M_u^T(\tilde{m})YM_u(\tilde{m}))) + \epsilon_A(\tilde{m})\min(\lambda_{\min}(Y), 0)$, thus $S_{\text{check}} > 0$ only if

$$\min(\text{eigs}(M_u^T(\tilde{m})YM_u(\tilde{m}))) + \epsilon_A(\tilde{m})\min(\lambda_{\min}(Y), 0) > 0,$$

for some $\tilde{m} > m$. By Lemma 6.2.7 this holds only if Y satisfies $Y \bullet M \geq 0$ for $M \in \partial f_{\lambda_2}(L)$ with $L \in [A, B]_{\text{LAP}}$. □

6.2.4 Information dissemination of robot positions

In order to execute Direction Checking Algorithm, robots first need information about the past states of the network to come up with reasonable bounds on the Laplacian matrix. Before specifying the protocol to disseminate information about each node throughout the network, we first address what it means for each node to hold information which is consistent with the real world. A formal definition is given next.

Definition 6.2.9 (Consistency of stored network information)**.** *Let $\mathcal{P}_{truth} \in \mathbb{R}^{n\times n}$ be the actual position of the robots at time t_{curr}, and let v_{max} be a bound on the maximum velocity of each individual robot. A tuple, $(\mathcal{P}, T, D)$, $\mathcal{P} \in \mathbb{R}^{d\times n}, T \in \mathbb{R}^n, D \in \mathbb{R}^{n\times n}$, is called* consistent *with $\mathcal{P}_{truth}$ at time t_{curr} if the following hold:*

(i) For each $i \in \{1, \ldots, n\}$, $\mathcal{P}_i \in B(\mathcal{P}_{truthi}, (t_{curr} - T_i)v_{max})$.

(ii) For each $i, j \in \{1, \ldots, n\} \times \{1, \ldots, n\}$, $\|\mathcal{P}_{truthi} - \mathcal{P}_{truthj}\| \in [D_{i,j} - v_{max}(t_{curr} - T_i + t_{curr} - T_j), D_{i,j} + v_{max}(t_{curr} - T_i + t_{curr} - T_j)]$. •

In other words, a set of information is CONSISTENT with an actual state of the world, (i) if the position of each robot, i, in the actual state of the world is within the range it could have reached by traveling with speed $v_{\max}$ starting from position $\mathcal{P}_i$ for time $t_{\text{curr}} - T_i$ and (ii) $D_{i,j}$ stores the distance between $\mathcal{P}_i$ and $\mathcal{P}_j$ and relates to the actual distance in the natural way.

Next we provide an algorithm which correctly disseminates CONSISTENT information on the state of the network to all robots. We start with the bookkeeping data necessary for this task. Each robot $i \in \{1, \ldots, n\}$ holds the following data structures

- For each other robot, j, a position, $\text{pos}^{[i]}(j) \in \mathbb{R}^d$ and a time $T_j^{[i]} \in \mathbb{R}$ since it was last known that the position of j was $\text{pos}^{[i]}(j)$.
- For each other robot, j, $S_{k,j}(i) \in \{\text{true}, \text{false}\}$ is true if and only if the most recent copy of $(\text{pos}^{[i]}(k), T_k^{[i]})$ needs to be sent from i to j.

The ALL-TO-ALL BROADCAST ALGORITHM is described in Table 6.2.

Note that, in lines `7:-8:` of ALL-TO-ALL BROADCAST ALGORITHM, each robot pushes its own updated information in its queue, as well as that of a randomly selected robot, at each communication round. While this is not necessary for our proof, in practice it leads to much better bounds on the instantaneous rates of change of the Laplacian due to individual robot motions, which primarily depend on accurate information on each agent's immediate neighbors.

Because this algorithm is randomized, we discuss its expected performance. In the following result, let $\text{path}_{j,k}(t)$ denote the shortest path between robots j and k at time t.

Name:	All-to-all Broadcast Algorithm
Goal:	Disseminate information about robot positions throughout the network
Inputs:	• $t_{\text{curr}} \in \mathbb{R}$ indicating the current time
Messages from neighbors:	• $j \in \{1, \ldots, n\}$ is identifier of the robot from which the signal originated • $t_{\text{valid}} \in \mathbb{R}$ indicating the time at which message from j originated • $P_j \in \mathbb{R}^d$, the position of j at time t_{valid}
Sensor Data	• $P_{\text{id}} \in \mathbb{R}^d$, current position of this robot, acquired via sensing
Persistent data:	• id $\in \{1, \ldots, n\}$, current robot's unique identifier • $T \in \mathbb{R}^n$, array of last recorded time information • $\mathcal{P} \in \mathbb{R}^{d \times n}$, array of last recorded position information • $S \in \{\text{true}, \text{false}\}^{n \times n}$ where $S_{i,j}$ indicates whether the most up-to-date information about i needs to be sent to j (value true) or not (value false) • $D \in \mathbb{R}^{n \times n}$, matrix of approximate inter-robot distances

Let $\mathcal{P}_{\text{id}} \leftarrow P_{\text{id}}$ and $T_{\text{id}} \leftarrow t_{\text{curr}}$ {*Update self*}
for all $m \in \{1, \ldots, n\} \setminus \{\text{id}\}$ **do**
  Let $S_{\text{id},m} \leftarrow$ true {*Sending phase*}
end for
for all $k \in \mathcal{N}$ **do**
  Randomly select j from elements having $S_{j,k} =$ true
  Push $(j, T_j, \mathcal{P}_j)$ onto the queue of items to be sent to agent k
  Push $(\text{id}, T_{\text{id}}, \mathcal{P}_{\text{id}})$ onto the queue of items to be sent to agent k
end for
Send two items from each queue to corresponding destination
for all $k \in \mathcal{N}$ **do**
  for all message $\{1, 2\}$ {*Each neighbor sent two messages previously*} **do**
    Receive j, t_{valid}, P_j from k. {*Receiving phase*}
    if $t_{\text{valid}} > T_j$ **then**
      for all $m \in \{1, \ldots, n\} \setminus \{\text{id}\}$ **do**
        Let $S_{j,m} \leftarrow$ true
      end for
      Let $S_{j,k} \leftarrow$ false, $T_j \leftarrow t_{\text{valid}}$, and $\mathcal{P}_j \leftarrow P_j$
      for all $i \in \{1, \ldots, n\} \setminus \{j\}$ **do**
        Let $D_{i,j} \leftarrow \|\mathcal{P}_i - \mathcal{P}_j\|$ and $D_{j,i} \leftarrow \|\mathcal{P}_i - \mathcal{P}_j\|$
      end for
    end if
  end for
end for

Table 6.2: All-to-all Broadcast Algorithm.

Lemma 6.2.10. *For $j \in \{1,\ldots,n\}$, let $k \in \{1,\ldots,n\}$ be the robot that maximizes $t_{curr} - T_j^{[k]}$ at time t_{curr}, in other words, k's estimate of j is the most out of date estimate on the network. If the communication graph is connected between rounds at time t and $t+\delta T$, then the expectation of $\sum_{i \in path_{j,k}(t)} T_j^{[i]}$, increases by at least $\frac{1}{n}|T_j^{[k]} - T_j^{[j]}|$ between t and $t+\delta T$ for any $j \in \{1,\ldots,n\}$. Likewise, if t_{curr} is the current time, the expectation of $\sum_{i \in path_{j,k}(t)}(t_{curr} - T_j^{[i]})$ (the sum of the amount by which the timestamps for j are out of date along the path from j to k) decreases by at least $\frac{1}{n}\max_{k,l \in \{1,\ldots,n\}}(|T_j^{[k]} - T_j^{[l]}|) - |path_{j,k}(t)|\delta T$ between t and $t+\delta T$ for any $j \in \{1,\ldots,n\}$.* •

Proof. Let i_{less} and i_{greater} be the two agent identities adjacent to an edge, e, having $T_j^{[i_{\text{less}}]} \leq T_j^{[i_{\text{greater}}]}$. With probability at least $\frac{1}{n}$ agent i_{greater} will broadcast its estimate of j to agent i_{less} increasing $T_j^{[i_{\text{less}}]}$ by $|T_j^{[i_{\text{greater}}]} - T_j^{[i_{\text{less}}]}|$. Broadcasting j the other direction does not affect any agents state estimate. Since $\mathcal{G}(\mathcal{P}(T))$ is connected for $T \in [t, t+\delta T]$, there must be at least one path between the agents with the least and greatest value of $T_j^{[i]}$, and a set of edges must exist along this path for which the sum of the differences of $T_j^{[i_{\text{greater}}]} - T_j^{[i_{\text{less}}]}$ must exceed $\max_{k \in \{1,\ldots,n\}}(|T_j^{[k]} - T_j^{[j]}|)$. The second half of the statement follows from the first. □

The next result characterizes the expected time by which the information held by each node may be out of date.

Corollary 6.2.11. *For any robot, $j \in \{1,\ldots,n\}$, the average expectation, over all robots, $i \in \{1,\ldots,n\}$ of $(t_{curr} - T_j^{[i]})$ never exceeds $(n^2+1)\delta T$. Likewise the expected maximum, over all $i \in \{1,\ldots,n\}$ of $t_{curr} - T_j^{[i]}$ never exceeds $n(n^2+1)\delta T$, and the expected maximum, over all $i,j \in \{1,\ldots,n\}$ of $t_{curr} - T_j^{[i]}$ never exceeds $n(n^2+1)$* •

Proof. By Lemma 6.2.10, $(\frac{1}{n}\sum_{i\in 1,\ldots n}(t_{\text{curr}} - T_j^{[i]}))$ decreases whenever

$$\frac{1}{n}\max_{k,l\in\{1,\ldots,n\}} |T_j^{[k]} - T_j^{[l]}| - n\delta T > 0.$$

Since at least one node (j) has the fresh value of j ($T_j^{[j]} = t_{\text{curr}}$), the expectation of $\max_{k,l\in\{1,\ldots,n\}}(|T_j^{[k]} - T_j^{[l]}|)$ is at least the average (over i) of the expectation of $t_{\text{curr}} - T_j^{[i]}$ and the average expectation of $t_{\text{curr}} - T_j^{[i]}$ decreases whenever the expectation of $\max_{k,l\in\{1,\ldots,n\}}(|T_j^{[k]} - T_j^{[l]}|)$ is above $n^2\delta T$. The expectation of the average over i of $t_{\text{curr}} - T_j^{[i]}$, being a lower bound of $\max_{k,l\in\{1,\ldots,n\}}(|T_j^{[k]} - T_j^{[l]}|)$, must decrease if it exceeds $n^2\delta T$. This quantity goes up by at most δT each round and thus never exceeds $n^2\delta T + \delta T$. Since the expectation of $t_{\text{curr}} - T_j^{[i]}$ is never below 0, the expectation of the maximum must be at most n times the average, thus it never exceeds $n(n^2+1)\delta T$.

For the last part of the statement, the average over all i of the expectation of $t_{\text{curr}} - T_j^{[i]}$ never exceeds $(n^2+1)\delta T$ for any fixed j, and so the average over all i, j of the expectation of $t_{\text{curr}} - T_j^{[i]}$ also never exceeds $(n^2+1)\delta T$. Since the maximum over all $i, j \in \{1,\ldots,n\}$ of $t_{\text{curr}} - T_j^{[i]}$ is at most n^2 times the average over all i, j of $t_{\text{curr}} - T_j^{[i]}$, this value never exceeds $n^2(n^2+1)\delta T$. □

Finally, we establish that the information stored by the network is consistent with the actual robot positions in the sense of Definition 6.2.9.

Theorem 6.2.12. *Assume each robot moves with velocity at most v_{max}. At all times, each robot holds values of $T, \mathcal{P}, D$ which are* CONSISTENT*, in the sense of Definition 6.2.9, with the state of the network at time t_{curr}.* •

Proof. Each broadcast message, as sent in ALL-TO-ALL BROADCAST ALGORITHM, contains a position of a robot, and the time at which that position was valid. Since the time and the position are both stored, the results always verify condition 1 of Definition 6.2.9. Condition 2 of Definition 6.2.9 holds

for any $(\mathcal{P}, T, D)$ where condition 1 holds for $\mathcal{P}$ and T, and $D_{i,j} = \|\mathcal{P}_i - \mathcal{P}_j\|$ for all i and j. $\square$

6.3 Algorithmic solutions to the connectivity problems

In this section, we combine the algorithmic procedures developed in Section 6.2 to decide if a proposed network motion is safe for connectivity maintenance and to disseminate position information across the network. This allows us to synthesize the Motion Test Algorithm and the Motion Projection Algorithm to solve the Spectral Connectivity Decision Problem and the Spectral Connectivity Problem, respectively.

6.3.1 Motion Test Algorithm

Here we combine the Direction Checking Algorithm and the All-to-all Broadcast Algorithm to synthesize a motion coordination algorithm that solves the Spectral Connectivity Decision Problem. First, we provide the Matrix Bound Generator in Table 6.3 to compute lower $A \in \mathrm{LAP}(n)$ and upper $B \in \mathrm{LAP}(n)$ bounds on the Laplacian matrix of the communication graph from the data disseminated via the All-to-all Broadcast Algorithm.

Lemma 6.3.1. *Let t_{curr} be the current time. Given a proximity graph induced by a monotonic function $g_{wgt} : \mathbb{R} \to \mathbb{R}$, for $(\mathcal{P}, T, D)$* consistent *with the current set of robot positions,* Matrix Bound Generator *returns two matrices which bound the Laplacian of the graph induced by $\mathcal{P}_{truth}$ at any time between t_{curr} and $t_{curr} + \delta T$.* •

Proof. Let $t \in [t_{\mathrm{curr}}, t_{\mathrm{curr}} + \delta T]$. By monotonicity of g_{wgt} and consistency of $(\mathcal{P}, T, D)$, the distance between i and j is between $D_{i,j} - v_{\max}((t_{\mathrm{curr}} - T_i + \delta T) + (t_{\mathrm{curr}} - T_j + \delta T))$ and $D_{i,j} + v_{\max}((t_{\mathrm{curr}} - T_i) + (t_{\mathrm{curr}} - T_j + \delta T))$, yielding the (i,j)-th element of the adjacency matrix in the interval $[g_{\mathrm{wgt}}(D_{i,j} - v_{\max}((t_{\mathrm{curr}} - T_i + \delta T) + (t_{\mathrm{curr}} - T_j + \delta T))), g_{\mathrm{wgt}}(D_{i,j} - v_{\max}((t_{\mathrm{curr}} - T_i + \delta T) + (t_{\mathrm{curr}} - T_j + \delta T)))]$. Since all the off-diagonal elements of the Laplacian matrices, A and B, are defined this way, and the diagonal

Name:	MATRIX BOUND GENERATOR
Goal:	Compute bounds $A, B \in \text{LAP}(n)$ of Laplacian of communication graph
Inputs:	• Current time $t_{\text{curr}} \in \mathbb{R}$ • Maximum velocity of any robot, $v_{\max}$ • Maximum time between communication rounds, δT
Persistent data:	• $T \in \mathbb{R}^n$, array of last recorded time information • $\mathcal{P} \in \mathbb{R}^{d \times n}$, array of last recorded position information • $D \in \mathbb{R}^{n \times n}$, matrix of approximate inter-robot distances
Outputs:	$A, B \in \text{LAP}(n)$, are the matrix bounds

1: **for all** $i \in \{1, \ldots, n\}$ **do**
2: **for all** $j \in \{1, \ldots, n\}$ **do**
3: Let $A_{i,j} \leftarrow g_{\text{wgt}}(D_{i,j} - v_{\max}((t_{\text{curr}} - T_i + \delta T) + (t_{\text{curr}} - T_j + \delta T)))$
4: Let $B_{i,j} \leftarrow g_{\text{wgt}}(D_{i,j} + v_{\max}((t_{\text{curr}} - T_i + \delta T) + (t_{\text{curr}} - T_j + \delta T)))$
5: **end for**
6: **end for**
7: Let $A \leftarrow \text{diag}(\mathbf{1}^T A) - A$
8: Let $B \leftarrow \text{diag}(\mathbf{1}^T B) - B$

Table 6.3: MATRIX BOUND GENERATOR.

elements are CONSISTENT with the definition of $\text{LAP}_{\pm}(n)$, the actual Laplacian matrix of the graph is in $[A, B]_{\text{LAP}}$ for all $t \in [t_{\text{curr}}, t_{\text{curr}} + \delta T]$. □

Next, we characterize the expected gap between the off-diagonal elements of A and B generated by MATRIX BOUND GENERATOR.

Lemma 6.3.2. *Let $g_{max} = \max_{x \in \mathbb{R}_{\geq 0}}(|g'_{wgt}(x)|)$. For $i \neq j \in \{1, \ldots, n\}$, the expected gap, $B_{i,j} - A_{i,j}$ never exceeds $4g_{max}v_{max}(n(n^2+2)\delta T)$. The expected maximum over $i \neq j, i \in \{1, \ldots, n\}, j \in \{1, \ldots, n\}$ $B_{i,j} - A_{i,j}$ never exceeds $4g_{max}v_{max}(n^2(n^2+2)\delta T)$.* •

Proof. The expectations of $t_{\text{curr}} - T_i$ and $t_{\text{curr}} - T_j$ never exceed $(n(n^2+1)\delta T)$ by Corollary 6.2.11. Since $A_{i,j} = g_{\text{wgt}}(D_{i,j} - v_{\max}((t_{\text{curr}} - T_i + \delta T) + (t_{\text{curr}} - T_j + \delta T)))$ and $B_{i,j} = g_{\text{wgt}}(D_{i,j} - v_{\max}((t_{\text{curr}} - T_i + \delta T) + (t_{\text{curr}} - T_j + \delta T)))$, their difference never exceeds $2g_{\max}v_{\max}(n(n^2+1+1)\delta T + n(n^2+1+1)\delta T)$.

By similar reasoning, the expected maximum over all $i \neq j$ of $B_{i,j} - A_{i,j}$ never exceeds

$4g_{\max}v_{\max}(n^2(n^2+1+1)\delta T)$ since this is at most twice $2g_{\max}v_{\max}$ times $\max_{i\in\{1,\ldots,n\}}(t_{\text{curr}}-T_i+\delta T)$. □

Finally, we combine the Direction Checking Algorithm which verifies winning solutions to Graph Picking Game with the All-to-all Broadcast Algorithm and the Matrix Bound Generator which provide position information to the network robots. This combination allows us to synthesize a solution to Spectral Connectivity Decision Problem. This solution, Motion Test Algorithm, is presented in Table 6.4. The next result shows that Motion Test Algorithm returns a value of $f_{\text{safe}} \geq 0$ only if the instantaneous change in the Laplacian due to motion in direction v wins Graph Picking Game.

Theorem 6.3.3. *Assuming that each robot moves with velocity at most v_{max},* Motion Test Algorithm *solves* Spectral Connectivity Decision Problem. •

Proof. By Theorem 6.2.12 the information received by each robot is consistent with the state of the network. Lemma 6.3.1 shows, given consistent data, that the matrices A and B properly bound the graph Laplacian of the communication graph. The motion in matrix space induced by the proposed instantaneous motion in $\mathbb{R}^{d\times n}$ is bounded from below by X_{lower}, i.e., it is a member of $\{Y \in \text{LAP}_{\pm}(n) \ : \ Y \geq_{\text{LAP}} X_{\text{lower}}\}$. Finally Theorem 6.2.8 and Corollary 6.2.4 show that Direction Checking Algorithm called on line `14:` of Motion Test Algorithm returns $f_{\text{safe}} \leq 0$ only when the proposed direction of motion wins Graph Picking Game, and hence is allowable under Spectral Connectivity Decision Problem. □

The next result shows that solutions to the Spectral Connectivity Decision Problem keep the algebraic connectivity of the network above the desired threshold.

Corollary 6.3.4. *If each robot runs an algorithm which solves* Spectral Connectivity Decision Problem, *then the algebraic connectivity λ_2 of the network never drops below λ_+.* •

Name:	Motion Test Algorithm
Goal:	Solve Spectral Connectivity Decision Problem.
Inputs:	• Current time $t_{\text{curr}} \in \mathbb{R}$ • Maximum velocity of any robot, $v_{\max}$ • Maximum time between communication rounds, δT • Proposed direction of motion, v • Eigenvalue bounds $\lambda_- \leq \lambda_+ \in \mathbb{R}$
Persistent data:	• $T \in \mathbb{R}^n$, an array of last recorded time information • $\mathcal{P} \in \mathbb{R}^{d \times n}$, an array of last recorded position information • $D \in \mathbb{R}^{n \times n}$, a matrix of rough inter-robot distances • $A, B \in \text{LAP}(n)$ • id $\in \{1, \ldots, n\}$, unique identifier of current robot
Outputs:	• $f_{\text{safe}} \in \mathbb{R}$ such that $f_{\text{safe}} \geq 0$ if, for any time $t \in [t_{\text{curr}}, t_{\text{curr}} + \delta T]$, the instantaneous change in the Laplacian matrix due to motion in the direction v wins Graph Picking Game at time t

Initialize $X_{\text{upper}} \leftarrow \mathbf{0}$
Initialize $X_{\text{lower}} \leftarrow \mathbf{0}$
for all $i \in \{1, \ldots, n\}$ **do**
  $X_{\text{lower}_{\text{id},i}} \leftarrow -\min_{p \in B(\mathcal{P}_i, v_{\max}(t - T_i + \delta T))}(g'_{\text{wgt}}(p, \mathcal{P}_{\text{id}}; v, \mathbf{0})$ {*Compute bounds on direction matrix*}
  Let $X_{\text{upper}_{\text{id},i}} \leftarrow -\max_{p \in B(\mathcal{P}_i, v_{\max}(t - T_i + \delta T))}(g'_{\text{wgt}}(p, \mathcal{P}_{\text{id}}; v, \mathbf{0})$
  Let $X_{\text{upper}_{\text{id,id}}} \leftarrow X_{\text{upper}_{\text{id,id}}} - X_{\text{upper}_{\text{id},i}}$
  Let $X_{\text{lower}_{\text{id,id}}} \leftarrow X_{\text{lower}_{\text{id,id}}} - X_{\text{lower}_{\text{id},i}}$
end for
Let $\lambda_- \leftarrow \max(\lambda_-, \lambda_2(A))$
Let $\lambda_+ \leftarrow \min(\lambda_+, \lambda_2(B))$
if $\lambda_- \geq \lambda_+$ **then**
  return 0 {*There are no possible matrices with eigenvalues in the disallowed range*}
end if
Let $f_{\text{safe}} \leftarrow$ Direction Checking Algorithm on $A, B, X_{\text{lower}}, \lambda_-, \lambda_+$
return f_{safe}

Table 6.4: Motion Test Algorithm.

Proof. Since each robot's individual motion solves Spectral Connectivity Decision Problem, whenever $\lambda_2 \leq \lambda_+$, we know $f^{\circ}_{2-\text{conn}}(\mathcal{P}(t);[\mathbf{0},\ldots,u_i^T,\ldots,\mathbf{0}]^T) \geq 0$, for $i \in \{1,\ldots,n\}$, for all times t. Since $[u_1,\ldots,u_n]^T = \sum_{i=1}^n [\mathbf{0},\ldots,u_i^T,\ldots,\mathbf{0}]^T$, by [11, Proposition 2.3.3],

$$f^{\circ}_{2-\text{conn}}(\mathcal{P}(t);[u_1,\ldots,u_n]^T) \subseteq [\sum_{i=1}^{n} \min(f^{\circ}_{2-\text{conn}}(\mathcal{P}(t);[\mathbf{0},\ldots,u_i^T,\ldots,\mathbf{0}]^T)), \sum_{i=1}^{n} \max(f^{\circ}_{2-\text{conn}}(\mathcal{P}(t);[\mathbf{0},\ldots,u_i^T,\ldots,\mathbf{0}]^T))],$$

and thus $f^{\circ}_{2-\text{conn}}(\mathcal{P}(t);[u_1,\ldots,u_n]^T) \geq 0$. □

6.3.2 Analysis of Motion Test Algorithm under perfect information

We wish to show that Motion Test Algorithm exhibits reasonable behavior as δT becomes small. To do so, we compare it to an idealized variant of Motion Test Algorithm under which each robot has perfect information.

We let Idealized Motion Test Algorithm be the algorithm defined by executing Motion Test Algorithm in continuous time, with $\delta T = 0$, and with perfect information about the state of the network available to each robot. We expound on how this is an idealized variant of Motion Test Algorithm in Lemma 6.3.5, which shows that Idealized Motion Test Algorithm allows any collective motion such that no individual robot's motion instantaneously decreases λ_2 unless $\lambda_2 > \lambda_+$.

Lemma 6.3.5. *Under* Idealized Motion Test Algorithm*, a direction proposed by robot j is accepted if and only if it does not decrease λ_2 when taken by itself or if $\lambda_2 > \lambda_+$.*

Proof. Consider a direction of motion, u_j, which induces an instantaneous rate of change, $X \in \text{LAP}_{\pm}(n)$ of the Laplacian matrix $L \in \text{LAP}(n)$. Consider the case when $X \bullet (vv^T) < 0$ for some $vv^T \in \delta\lambda_2(L)$ and $\lambda_+ \geq \lambda_2(L)$. Line 1: of Direction Checking Algorithm ensures that λ_2 is used in place of

λ_+ in picking $m_{\min}$. Since each such (vv^T) satisfies $Lv = \lambda_2(L)$, $v \in \mathbf{u}_{\text{span-}L}(m)$ for any m having $\lambda_{m+1}(L) > \lambda_2(L)$. Thus the eigenvector calculation in line `17:` of DIRECTION CHECKING ALGORITHM returns a value less than 0 and the proposed motion is rejected. If $\lambda_+ \geq \lambda_2(L)$ and the direction, X, does not satisfy $X \bullet (vv^T) < 0$ for any $vv^T \in \delta\lambda_2(L)$, then there is no basis element, u, in $\mathbf{u}_{\text{span-}L}(m_{\min} - 1)$ having $u^T X u < 0$, so there is at least one m which produces a value greater than or equal to zero in line `17:` of DIRECTION CHECKING ALGORITHM. In the case in which $\lambda_+ < \lambda_2(L)$, lines `3:` and `4:` of DIRECTION CHECKING ALGORITHM force the direction to be allowed. □

The following lemma establishes various useful facts that hold with high probability as δT approaches zero.

Lemma 6.3.6. *If the second derivative of g_{wgt} is bounded, and g_{wgt} has zero derivative at 0, then for any configuration, for any $k \in \mathbb{R}$ and any proposed physical motion of robot j, u_j, there exists a time step δT such that, with high probability, the following are true:*

(i) The actual Laplacian, $L \in LAP(n)$ is within k of the estimated lower and upper bounds on the Laplacian ($A, B \in LAP(n)$), i.e., $\|A - L\|_2 < k$ and $\|B - L\|_2 < k$.

(ii) The actual instantaneous direction of motion, $Y \in LAP_{\pm}(n)$, is within k of the lower bound on the direction of motion, $X \in LAP_{\pm}(n)$, i.e., $\|Y - X\|_2 < k$.

(iii) For some m, $\epsilon_A(m) < k$.

Proof. Fact (i) follows from Lemma 6.3.2. Corollary 6.2.11 allows us to bound the expected time by which the information robot i holds about robot j is out of date. The bound is a decreasing function of δT. This bound is linearly related to the bound on the radius of a sphere known to contain robot j, which induce decreasing bounds on both the range of angles i can be relative to j and the range

of distances i can be from j. If the second derivative of g_{wgt} is also bounded, these induce a bound on the computations in lines `4:` and `5:` of MATRIX BOUND GENERATOR. Since the results of these computations form upper and lower bounds for Y, we can deduce that the distance from the lower bound to Y can be bounded by a decreasing function of δT, thus showing fact (ii). Regarding fact (iii), note that $g'_{\text{wgt}}(0) = 0$, thus the proximity graph is smooth with bounded second derivative even where two robots are coincident. Assume $\lambda_n(L) \neq \lambda_2(L)$. Let l be the index such that $\lambda_l(A) > \lambda_{l-1}(A) = \lambda_2(A)$. From part 1, δT can be picked such that, with high probability, $\|(B - L)\|_2 < \lambda_+ - \lambda_2(L)$. Thus we can replace the expression in line `11:` of DIRECTION CHECKING ALGORITHM with $\sqrt{\frac{\lambda_2(B)-\lambda_2(A)}{\lambda_l(A)-\lambda_2(A)}}$ There exists a δT such that the maximum eigenvalue of the expected difference between A and B is less than any constant, thus allow $\epsilon_A(l)$ to be chosen to be less than any given constant. If $\lambda_n(L) = \lambda_2(L)$, let $l = n + 1$, and note that $\epsilon_A(n) = 0$. □

The next result shows that, as δT approaches zero, the behavior of MOTION TEST ALGORITHM approaches that of IDEALIZED MOTION TEST ALGORITHM.

Theorem 6.3.7. *For any configuration, and any proposed direction of motion, v, for robot j, which is permitted under* IDEALIZED MOTION TEST ALGORITHM*, there exists a time step, δT, such that when communication happens every δT time units, with high probability robot j will be allowed to move in direction v.* •

Proof. Let Y be the instantaneous change in the Laplacian matrix induced by v. Let $-3k > \min_{\{v\,:\,Lv=\lambda_2 v\}}(Y \bullet (vv^T))$ Let l be defined as in Lemma 6.3.6, i.e., $\lambda_l(L) > \lambda_{l-1}(L) = \lambda_2(L)$ or $l = n + 1$ and $\lambda_{l-1}(L) = \lambda_n(L)$. By Lemma 6.3.6 there is a δT such that $\epsilon_A(l-1) < k$ and $|\lambda_{\min} - \min(\text{eigs}(Y))| < k$. Likewise, by Lemma 6.3.6, we can pick X and A sufficiently close to Y and L respectively that $|(Y - X) \bullet A)| + |Y \bullet (L - A)| \leq k$ thus guaranteeing that $M_u^T((l-1))XM_u((l-1))(1 - \epsilon_A^2(l-1)) + \epsilon_A^2(l-1)\min(\lambda_{\min}, 0) \geq 0$. □

6.3.3 Motion Projection Algorithm

In this section, we introduce Motion Projection Algorithm to solve the Spectral Connectivity Problem, see Table 6.5. For this algorithm, we set the dimensionality of physical space, d, to be 2. In other words, each robot lives in $\mathbb{R}^2$. Roughly speaking, Motion Projection Algorithm is a root-finder procedure wrapped around Motion Test Algorithm. The idea is to find the minimum angle by which to deviate a proposed direction of motion so that Motion Test Algorithm returns $f_{\text{safe}} \geq 0$ when executed with the resulting projected direction. If this is the case, then Theorem 6.3.3 guarantees that the change in the Laplacian due to motion in the projected direction wins Graph Picking Game.

Let us start by introducing some useful notation. Given a vector $v = [v_1, v_2]^T \in \mathbb{R}^2$, let $f_{\text{motion}} : \mathbb{R} \mapsto \mathbb{R}$ be defined as follows: for each $\theta \in \mathbb{R}$, $f_{\text{motion}}(\theta)$ is the result of evaluating Motion Test Algorithm with the direction of motion $[v_1 \cos(\theta) - v_2 \sin(\theta), v_2 \cos(\theta) + v_1 \sin(\theta)]^T$. The following result provides an upper bound on how f_{motion} changes with θ.

Lemma 6.3.8. *f_{motion} is globally Lipschitz with Lipschitz constant $nv_{max} \max_{s\in\mathbb{R}}(g'_{wgt}(s))$.*

Proof. We begin by noting that $f_{\text{motion}}(\theta)$ is the minimum, over $\tilde{m} > m$, of

$$\min(\text{eigs}(M_u^T(\tilde{m}) X M_u(\tilde{m}))) + \epsilon_A(\tilde{m}),$$

where $M_u(\tilde{m})$ and $\epsilon_A(\tilde{m})$ do not depend on θ. Each off-diagonal element of X has a Lipschitz constant bounded by $v_{\max} \max_{s\in\mathbb{R}}(g'_{\text{wgt}}(s))$. From here on, let $v_{\text{edg}} = v_{\max} \max_{s\in\mathbb{R}}(g'_{\text{wgt}}(s))$. Let $L_{\text{clique}}(n) \in \text{LAP}(n)$ satisfy $L_{\text{clique}}(n)_{i,j} = -1$ for each $i \neq j$. The change in X due to a change in θ of $\Delta\theta$ lives in $\text{LAP}_{\pm}(n)$ and is bounded from above by $v_{\text{edg}} L_{\text{clique}}(n)\Delta\theta$ and from below by $-v_{\text{edg}} L_{\text{clique}}(n)\Delta\theta$. Let Δ*expression* be the change in the value of *expression* due to a change in θ of $\Delta\theta$. Since $M_u(\tilde{m})$

Name:	Motion Projection Algorithm
Goal:	Solve Spectral Connectivity Problem
Inputs:	• Current time $t_{\text{curr}} \in \mathbb{R}$
	• Maximum velocity of any robot, $v_{\max}$
	• Maximum time between communication rounds, δT
	• Proposed direction of motion, v
	• Eigenvalue bounds $\lambda_- \leq \lambda_+ \in \mathbb{R}$
Persistent	• $T \in \mathbb{R}^n$, an array of last recorded time information
data:	• $\mathcal{P} \in \mathbb{R}^{2\times n}$, an array of last recorded position information
	• $D \in \mathbb{R}^{n\times n}$, a matrix of approximate inter-robot distances
	• $A, B \in \text{LAP}(n)$
	• id $\in \{1, \ldots, n\}$, unique identifier of current robot
	• Maximum angle deflection, $\theta_{\max-\text{id}}$
	• $x_{\text{incr}} \in \mathbb{R}$, step-size for root finder.
Outputs:	• $\tilde{v} \in \mathbb{R}^2$, safe projected direction.
	• $\theta \in S_1$, angle by which to rotate v to get safe direction

---

Let $D \leftarrow$ call All-to-all Broadcast Algorithm on t_{curr}
Let $(A, B) \leftarrow$ Matrix Bound Generator on t_{curr}, $v_{\max}$ and δT
Let $v_\perp \leftarrow [-v_1, v_2]^T$ and $\theta \leftarrow 0$ {*Perpendicular direction, for computing rotations*}
while $\theta \leq \theta_{\max-\text{id}}$ **do**
  for all $x_{\text{sgn}} \in \{-1, 1\}$ **do**
    Let $\tilde{v} \leftarrow v\cos(\theta) + x_{\text{sgn}} v_\perp \sin(\theta)$ {*Rotated direction*}
    Let $S_{\text{check}} \leftarrow$ Motion Test Algorithm on $t_{\text{curr}}, v_{\max}, \delta T, \tilde{v}$
    if $x_{\text{sgn}} = -1$ or $|S_{\text{check}}| < k$ **then**
      Let $k \leftarrow |S_{\text{check}}|$ {*For stepsize computation*}
    end if
    if $S_{\text{check}} \geq 0$ **then**
      return $(\tilde{v}, x_{\text{sgn}}\theta)$ {*Found good direction*}
    end if
  end for
  Let $\theta \leftarrow \theta + \max(x_{\text{incr}}, \frac{k}{nv_{\max} \max_{s\in\mathbb{R}}(g'_{\text{wgt}}(s))})$ {*Step* θ}
end while{*At this point, no safe direction has been found.*}
return $([0, 0]^T, 0)$

Table 6.5: Motion Projection Algorithm.

is defined by a subset of the columns of an orthogonal basis, each element of $\text{eigs}(M_u^T(\tilde{m})X)$ has a Lipschitz constant contained in

$$[-v_{\text{edg}} \max(\text{eigs}(L_{\text{clique}}(n))), v_{\text{edg}} \max(\text{eigs}(L_{\text{clique}}(n)))] = [-v_{\text{edg}}n, v_{\text{edg}}n].$$

Because

$$\min(\text{eigs}(A+B)) \in [\min(\text{eigs}(A) + \min(\text{eigs}(B)), \min(\text{eigs}(A)) + \max(\text{eigs}(B))],$$

we deduce that

$$\Delta \min(\text{eigs}(M_u^T(\tilde{m})XM_u(\tilde{m}))) \in [\min(\text{eigs}(M_u^T(\tilde{m})\Delta XM_u(\tilde{m}))), \max(\text{eigs}(M_u^T(\tilde{m})\Delta XM_u(\tilde{m})))]$$

and therefore $\Delta \min(\text{eigs}(M_u^T(\tilde{m})XM_u(\tilde{m}))) \in [-v_{\text{edg}}n, v_{\text{edg}}n]$. □

The following result establishes that the root finder embedded in Motion Projection Algorithm uses a reasonable step size.

Lemma 6.3.9. *Let $v_{edg} = v_{max} \max_{s\in\mathbb{R}}(g'_{wgt}(s))$. If $f_{motion}(\theta) < -k$ for some $k \in \mathbb{R}_{\geq 0}$, $f_{motion}(\theta + \frac{k}{nv_{edg}}) < 0$.* •

Proof. The bound that the Lipschitz constant for $f_{\text{motion}}(\theta)$ lies in $[-nv_{\text{edg}}, nv_{\text{edg}}]$ holds for all θ. So the change in f_{motion} from θ to $\theta + \frac{k}{nv_{\text{edg}}}$ is bounded within $\frac{k}{nv_{\text{edg}}}[-nv_{\text{edg}}, nv_{\text{edg}}]$. □

Finally, we are ready to show that Motion Projection Algorithm is a solution to Spectral Connectivity Problem.

Theorem 6.3.10. *The* Motion Projection Algorithm *solves the* Spectral Connectivity Problem. *If there is an interval,* $[\theta_-, \theta_+] \subseteq S_1$*, such that* $f_{motion}(\alpha) \geq 0$ *for all* $\alpha \in [\theta_-, \theta_+]$ *and* $\theta_+ - \theta_- \geq x_{incr}$ *then* Motion Projection Algorithm *will return a direction other than* $[0,0]^T$. •

Proof. The outer loop in Motion Projection Algorithm checks possible directions and evaluates Motion Test Algorithm on them. If one of these directions results in Motion Test Algorithm returning a non-negative value, Motion Projection Algorithm returns that direction, otherwise it returns $[0,0]^T$. Let the current angle be θ. If θ steps by $\frac{k}{nv_{\max} \max_{s\in\mathbb{R}}(g'_{\text{wgt}}(s))}$, where k is the max of f_{motion} over $\{\theta, -\theta\}$, then f_{motion} will be less than zero for the next θ. If θ steps by more than this, then θ steps by x_{incr}. To pass the boundaries of the region $[\theta_-, \theta_+]$, θ must step by x_{incr}, and must, therefore land in $[\theta_-, \theta_+]$. Because each step of θ is at least x_{incr}, θ covers the entire region from 0 to $\theta_{\max-\text{id}}$ in finite time. □

6.3.4 Analysis of the communication complexity

At each communication round, the above solutions to the Spectral Connectivity Decision Problem and the Spectral Connectivity Problem require each robot to perform computations whose memory requirements are polynomial in the number of robots and whose time complexity is polynomial in the number of robots times the time required to calculate the eigenspace of an $n \times n$ matrix.

As is typically the case with any coordination algortihm, multiple communication rounds are required in order to complete the assigned task. Therefore, it is also important to study the complexity of our algorithm in terms of the required rate of communication to achieve a desired performance. This is related, in some sense, to the notion of communication complexity, commonly discussed in the literature on distributed algorithms [7, 33, 44], in which one studies the number of messages that need

to be sent during an algorithm execution in order to achieve a given task, usually written as a function of the size of the network.

In addition to the size of the network, there are additional factors that need to be considered to accurately characterize the required rate of commmunication, including the required performance (represented here as the probability that an agent will move, $p_{\text{move}} \in (0,1)$), the amount we expect $\lambda_2(\mathcal{G})$ to be above the threshold λ_+, the maximum velocity with which the robots move, $v_{\max}$, the time between communication rounds, δT, and a bound, $g_{\max}$, on the magnitude of the gradient of the proximity function.

We begin by characterizing a property of the matrices A and B generated by the MATRIX BOUND GENERATOR which we will use to bound the gap between $\lambda_2(A)$ and $\lambda_2(L(\mathcal{G}))$.

Lemma 6.3.11. *For any $p_{move} \in (0,1)$, the following holds*

$$\Pr\Big(\max_{i,j\in\{1,\dots,n\}}(B_{i,j}-A_{i,j})\Big) \leq \frac{4}{1-p_{move}} g_{max} v_{max}(n^2(n^2+2)\delta T) \leq p_{move}. \qquad \bullet$$

Proof. Follows from Markov's inequality, $\Pr(|X| > \alpha) \leq \frac{E(|X|)}{\alpha}$ and Lemma 6.3.2. □

We next proceed to combine this result with some general properties of graph Laplacians to bound the transmission rate necessary for each robot to be allowed to move in any direction with high probability (and thus be allowed to move in the direction specified by the underlying control algorithm).

Theorem 6.3.12. *Let $g_{max} = \max_{x\in\mathbb{R}_{\geq 0}}(|g'_{wgt}(x)|)$. If the time between communication rounds is such that $\delta T \leq \frac{(1-p_{move})(\lambda_2(L(\mathcal{G}))-\lambda_+)}{4g_{max}v_{max}(n^2(n^2+2)}$, then each robot will move on each timestep with probability at least p_{move}.*

Proof. By Lemma 6.3.11, with probability at least p_{move}, $\max_{i\neq j, i\in\{1,\dots,n\}, j\in\{1,\dots,n\}} B_{i,j} - A_{i,j}$ never exceeds $\frac{4}{1-p_{\text{move}}} g_{\max} v_{\max}(n^2(n^2+2)\delta T)$. Thus, with probability at least p_{move}, $B-A$ satisfies $B-A \leq_{\text{LAP}}$

$\frac{4}{1-p_{\text{move}}} g_{\max} v_{\max} (n^2(n^2+2)\delta T) L_{\text{unit}}$ where $L_{\text{unit}} \in \text{LAP}(n)$ has off-diagonal entries consisting solely of 0 and 1. The maximum possible eigenvalue of such a matrix is n, as shown in [18], and $\lambda_2(B) \leq \lambda_2(A) + \lambda_n(B-A)$, thus $\lambda_2(B) - \lambda_2(A) \leq \lambda_2(B-A)$ Whenever $\lambda_2(L)$ satisfies $\lambda_2(L(\mathcal{G})) \geq \lambda_+ + \delta\lambda$, setting $\delta T \leq \frac{(1-p_{\text{move}})\delta\lambda}{4 g_{\max} v_{\max} (n^2(n^2+2)}$, or equivalently, setting the transmission rate to approximately $\frac{8}{1-p_{\text{move}}} g_{\max} v_{\max} n^4$ real numbers per time unit allows the robots to move at each timestep with probability at least p_{move}.

□

We note that this result does not fully characterize the communication complexity of our solutions because, among other things, does not account explicitly for the proposed direction of motion of the robots. We discuss this topic among our ideas for future research later in Section 6.5.

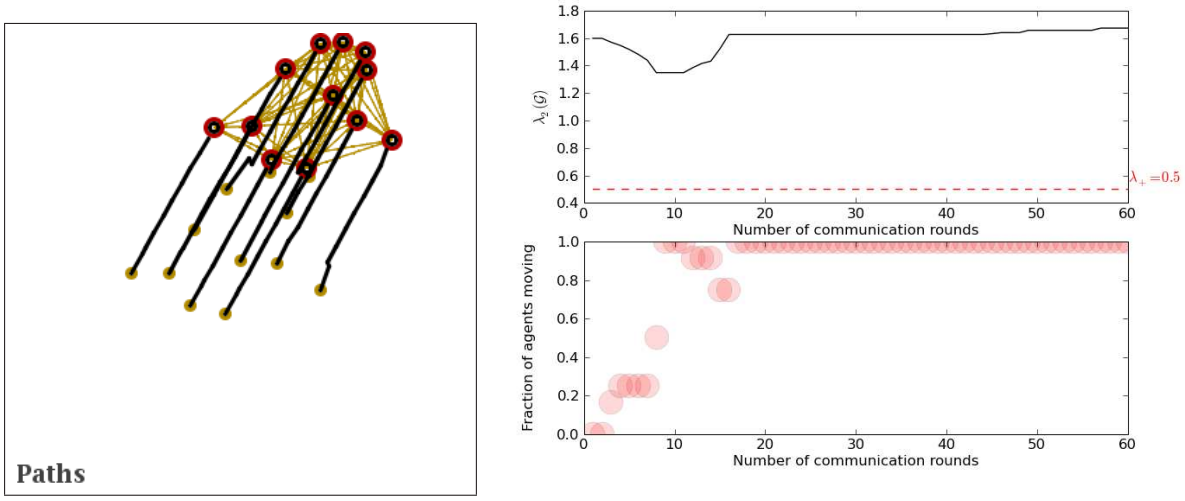
6.4 Simulations

In this section, we present simulations of the Motion Projection Algorithm to further validate the results presented in the previous sections. We have developed a custom Java-based platform [53] for the simulation of algorithms running on networks of robotic agents following the modeling framework proposed in [37]. The platform has a user interface layer which allows the simulations to be graphically displayed in real time in a Java applet.

In all simulations, Motion Projection Algorithm is implemented with the r-disk graph for the actual communication network, and the nonconvex weight function of the spline graph defined in Remark 2.4.2, where $r_{\max} \in \mathbb{R}$ was chosen to be slightly less than r and $r_{\min} \in \mathbb{R}$ was slightly bigger than zero. Note that the function g_{wgt} satisfies the conditions of Lemma 6.3.6. The maximum velocity of each individual robot is $v_{\max} = 0.125$ and the time step is $\delta T = 0.125$.

We consider four sets of underlying control laws. In the first simulation, shown in Figure 6.1,

all robots attempt to follow a Laplacian-based flocking algorithm. Each robot moves at unit speed and, at each time step, updates its heading to its own heading plus a scalar (0.1) times the average of the differences between its own heading and those of its neighbors. This is in fact a discrete-time implementation of the Laplacian-based averaging consensus [42], see also [23]. Each robot tries to move in the direction of its own heading, subject to maintaining connectivity.

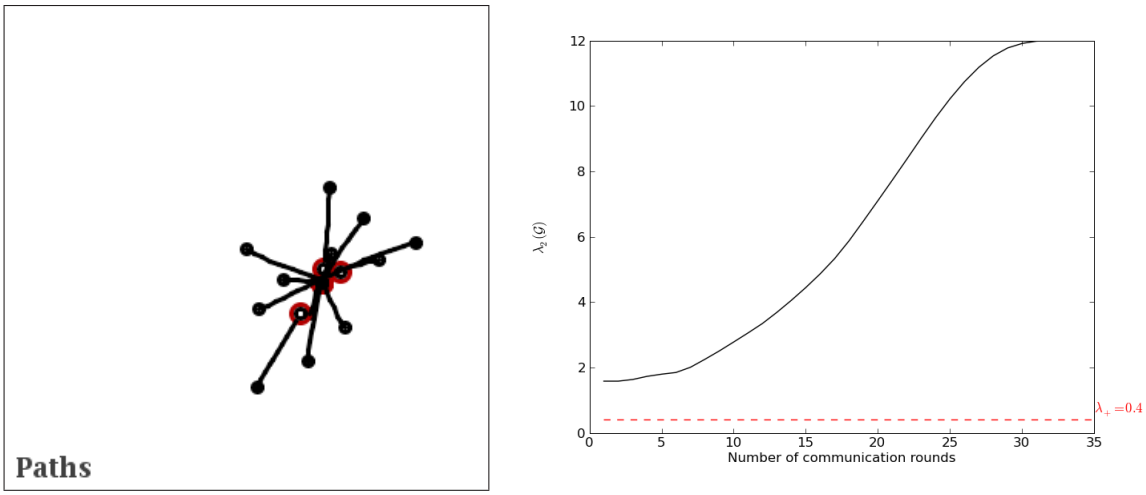


(a) (b)

Figure 6.1: Execution of Motion Projection Algorithm with 12 robotic agents. The underlying control law is determined by the robots moving at unit speed and running a Laplacian-based consensus to update its heading. Plot (a) shows the paths taken by the robots and plot (b) shows the evolution of the algebraic connectivity and the proportion of robots activel moving as functions of the communication round. The angle of motion each robot is allowed to deviate from is 0.95π.

In the second simulation, shown in Figure 6.2, all robots attempt to follow a Laplacian-based rendezvous algorithm. Each robot moves at unit speed and, at each time step, moves towards the average of its neighbors positions, while maintaining connectivity.

In the third simulation, shown in Figure 6.3, four agents with different conflicting control laws attempt to follow their own directives while maintaining connectivity. Note that the simulations in which the control directives naturally align with maintaining connectivity tend to require far fewer iterations to converge than those in which this is not the case (particularly those with random or



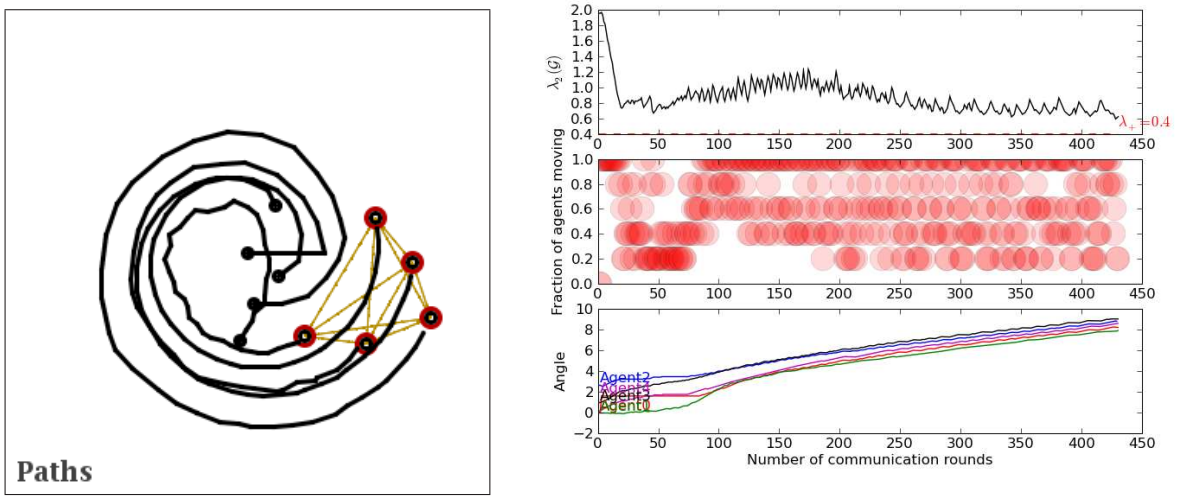
(a) (b)

Figure 6.2: Execution of Motion Projection Algorithm with 12 robotic agents. The underlying control law is determined by the robots moving at unit speed and running a Laplacian-based consensus to update its target position. Plot (a) shows the paths taken by the robots and plot (b) shows the evolution of the algebraic connectivity and the proportion of robots activel moving as functions of the communication round. The angle of motion each robot is allowed to deviate from is π.

conflicting motion).

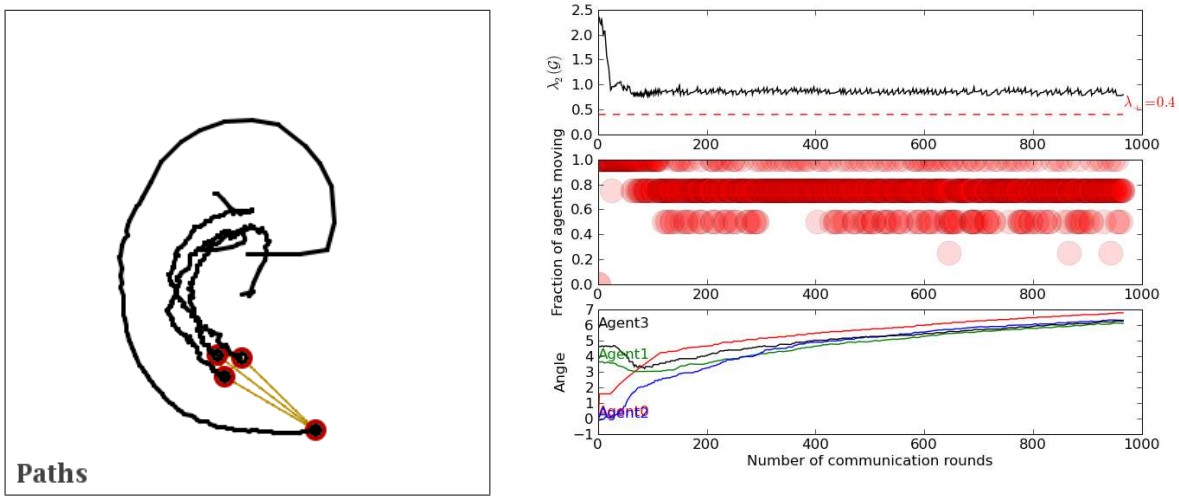
In the fourth simulation, shown in Figure 6.4, one leader robot attempts to follow a fixed trajectory while the remaining robots try to move randomly subject to the constraint of maintaining connectivity. For each robot, we specify the threshold $\theta_{\max-i} = 0.2$.

These simulations validate the preliminary results in Section 6.3.4 linking the communication complexity of the algorithm to the difference between λ_2 and λ_+. In particular, we observe that the fraction of agents moving at any given time is correlated to the difference between λ_2 and λ_+, and that the fraction of agents moving together with $v_{\max}$ affect the time required for the network to complete the given task. We also find that, in situations where λ_2 stays well above λ_+, the complexity is reasonable. On the other hand, the complexity degrades as the algorithm begins to push λ_2 up against the threshold. Adding a single agent whose underlying motion is random may help to maintain connectivity, but assigning random underlying motion to the majority of the agents, as in the simulation



(a) (b)

Figure 6.3: Execution of Motion Projection Algorithm with 4 robotic agents. The underlying control law corresponds to following scenario: Three leaders each attempt to follow different control laws each of which converges on a different fixed trajectory. The remaining agent moves randomly. Plot (a) shows the paths taken by the robots and plot (b) shows the evolution of the algebraic connectivity, the fraction of robots moving at each round, and the evolution of the angle of each robot's position relative to the origin.



(a) (b)

Figure 6.4: Execution of Motion Projection Algorithm with 5 robotic agents. The underlying control law corresponds to one leader following a fixed trajectory and the remaining agents moving randomly. Plot (a) shows the paths taken by the robots and plot (b) shows the evolution of the algebraic connectivity, the fraction of robots moving at each round, and the evolution of the angle of each robot's position relative to the origin.

shown in Figure 6.4, may slow down collective agent motion.

6.5 Conclusions

We have studied the problem of connectivity maintenance in robotic networks performing spatially-distributed tasks. In our approach, the edge weights of the connectivity graph are not necessarily convex functions of the inter-robot distances. We have proposed a distributed procedure to synthesize motion constraints on the individual robots so that the algebraic connectivity of the overall network remains above a desired threshold. The algorithm works even though individual robots only have partial information about the network state due to communication delays and network mobility. We have shown that as the communication rate increases, the performance of the proposed algorithm approaches that of the ideal centralized solution of Spectral Connectivity Decision Problem.

6.5.1 Comparison to other algorithms

Other distributed algorithms to control the algebraic connectivity of a spatially induced network includes those presented in [16, 67] and [69]. We discuss the relative merits of each approach here. The first such work to come to our attention, [16], operates by computing the gradient, $v_2 v_2^T$, of λ_2 in the space of Laplacian matrices. It them restricts robot motion to have the appropriate effect (growth or control) on λ_2 based on its gradient. Their approach, like ours, requires that each robot store $O(n^2)$ numbers while computing the gradient. Unlike our approach, theirs requires that each robot send multiple messages of size $O(n^2)$ per motion allowed by the gradient algorithm. It also restricts the inter-robot edge weights to be of a particular form based on decaying exponetials, unlike our restriction that the inter-robot edge weights be monotonically decreasing as a function of inter-robot distance and have zero derivative at special places. The multiple messages of $O(n^2)$ real numbers (or

approximations thereof) are sent within a numerical distributed averaging operation which, in turn, is executed multiple times within a numerical method (based on the QR algorithm from linear algebra) for finding the eigenvector v_2.

A simple improvement to this approach might be to come up with an upper bound, $\lambda_{\text{max-est}}$ for λ_n, and, instead of finding the top $n-1$ eigenvectors of the Laplacian matrix, L, instead find the top 2 eigenvectors of $\lambda_{\text{max-est}} I - L$.

This approach forms part of the series of improvements to this basic gradient algorithm proposed in [67]. Their algorithm requires robots to send messages of $O(1)$ real numbers and to store only $O(1)$ real numbers (while our algorithm requires each robot store $O(n^2)$ real numbers). The authors of [67] express their algorithm as a continuous-time algorithm, rather than a discrete-time communication algorithm. To rewrite it in our framework would require numerically discretizing their integrator-based approach. To date, no study has been done on the communication rate bounds required to do this effectively, and how they compare to those of our algorithm.

An approach which seems like a comprimise between the algorithms presented in this chapter and those in Chapter 5 is presented in [69]. Like our approach in Chapter 5 this work decouples the problem of constraining motion to preserve a link from that of deciding which links to preserve. Like the work in Chapter 6, each robot slowly propogates information about the entire graph and acts on locally available estimates of the state of the network to ensure the algebraic connectivity has desired properties. Unlike either of the above algorithms, the method proposed in [69] requires an auction to be held across the entire network for the deleting of each edge. It is currently unknown under what conditions this algorithm allows a link to be dropped while ours doesn't and vice versa.

Part IV

Concluding remarks

Chapter 7

Lessons from studies of multiple connectivity problems

Some lessons learned from these studies include the following:

- Synchronized operations help with connectivity algorithms.
- Special graphs can be useful
- It can be valuable to decouple a connectivity algorithm into a component which handles individual links and a component which decides which links to handle.
- While connectivity is a global property, there are multiple methods for transmitting global information, each of which yields connectivity algorithms with different properties

The Connectivity Maintenance Algorithm discussed in Chapter 5 is heavily dependent on the synchronized nature of the network model in which it is implemented. Many of the proofs of correctness fail if adjacent robots are on different communication rounds. A practical implementation of this algorithm would have to rely on some external (possibly expensive) synchronization method. Since

most tree rearrangements allowed by the algorithm happen in very few steps it might be reasonable to achieve this by running CM Algorithm at a very slow rate compared to the communication rate required by the underlying control law allowing rounds of CM Algorithm to be triggered by a slow simulation of synchronous communication on an otherwise asynchronous network. Likewise the algorithms described in Part II require that spacecraft simultaneously and syncrhonously start and end certain motion operations. If the robot motion speed is low relative to the communication rate or the drift of the robot's on-board clocks, this is feasible.

Special graphs can be useful for connectivity algorithms. The Connectivity Maintenance Algorithm relies on a tree as a certificate of graph connectivity. Likewise the algorithms presented in Part II use properties of the bipartite graph to reduce the problem of "graph connectivity" to one of link connectivity.

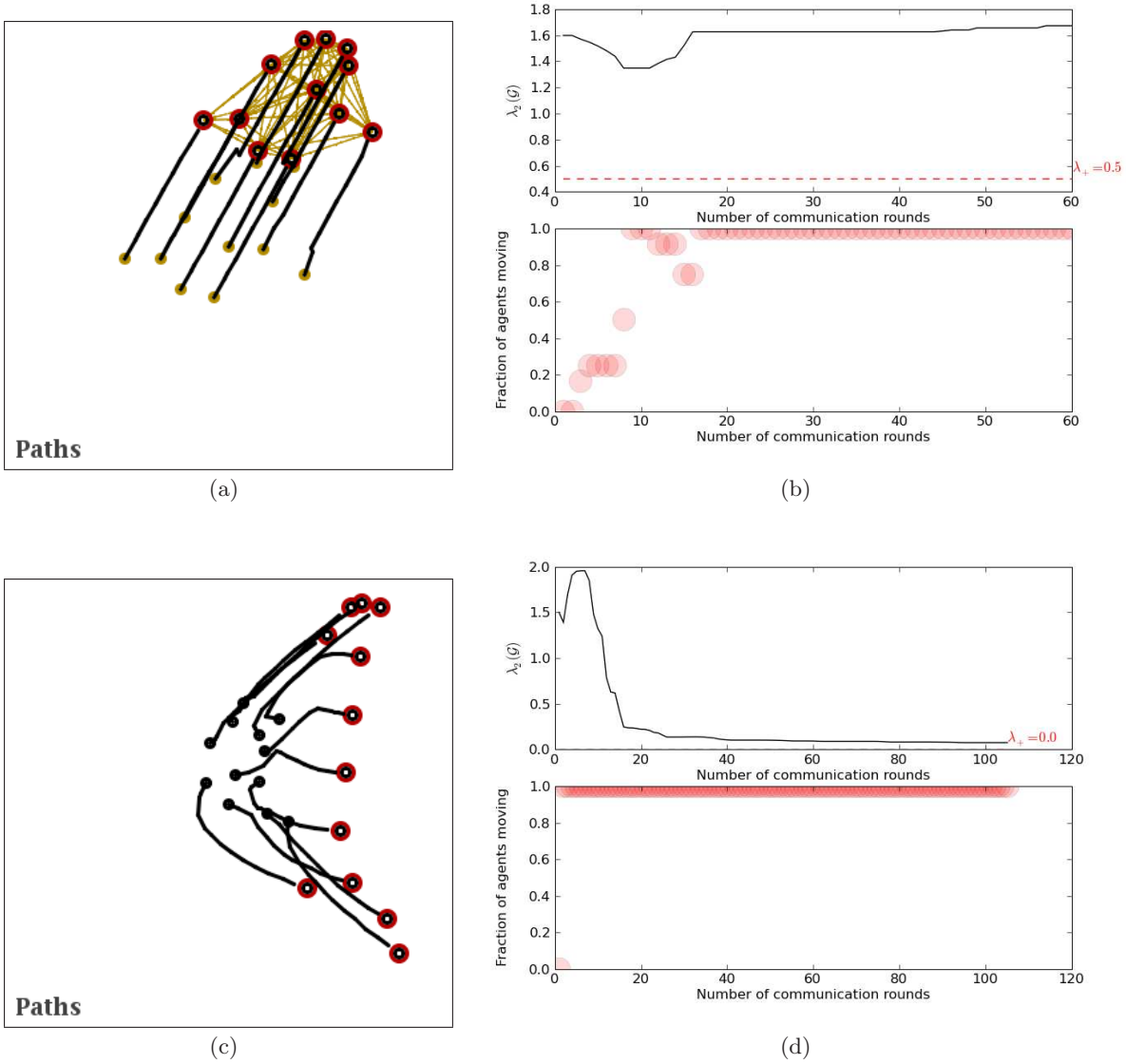
Relatedly, both of the algorithms discussed above decouple the problem into a component which deals with graph connectivity (whether it be negotiations to maintain a tree, or agreement on which agents will be in which component of the bipartite graph in Part II) and a component which handles individual links (preserving them in Chapter 5 and creating them in Part II).

Finally, different methods of transmitting global information yield different connectivity algorithms. Three ways of computing values which depend on the state of all nodes on a network are

- Explicitly perform an all-to-all broadcast, as in Chapter 6 and [69].
- Express the final computed value as a function of the sum of values computed at each node, and run a distributed averaging algorithm, much like the work presented in [16, 67]
- Express the final value as a heierarchical function and compute it over a tree, as in Chapter 5.

Each of these has been used in at least one connectivity maintenance algorithm, and each class of algorithms resulting from one of these methods has different properties. The tree method can compute final values in time proportional to the depth of the tree, but may have issues with synchronization and brittleness. The explicit broadcast algorithm easily handles asynchronous communication and lost messages, but at great cost in terms of communication complexity. The distributed averaging approach can represent a compromise between the two (if done properly).

As a final note, we directly compare Connectivity Maintenance Algorithm with Motion Projection Algorithm by showing the results of each algorithm coupled with a flocking algorithm in Figure 7.1.



(a) (b)

(c) (d)

Figure 7.1: Execution of Motion Projection Algorithm with 12 robotic agents in (a) and (b). The underlying control law is determined by the robots moving at unit speed and running a Laplacian-based consensus to update its heading. Plot (a) shows the paths taken by the robots and plot (b) shows the evolution of the algebraic connectivity and the proportion of robots activel moving as functions of the communication round. The angle of motion each robot is allowed to deviate from is 0.95π. Execution of CM Algorithm with 12 robotic agents in (c) and (d) with the same underlying control law. The agent velocity in (c) and (d) is set to half that in (a) and (b)

Chapter 8

Potential directions of future research

Potential future work connected with Chapter 5 includes understanding how the resulting trees of our algorithm compare to minimum spanning trees, developing systematic ways to encode preference re-arrangements in connection with other coordination algorithms and exploring the efficacy of the proposed ideas in conjunction with other proximity graphs relevant in motion coordination, such as the visibility graph.

Future work on the algorithms presented Chapter 6 could evaluate the communication complexity of the proposed coordination algorithms. We are especially interested in calculating lower bounds on the communication complexity required by the computation of the gradient of the algebraic connectivity function in a distributed way. We would also like to study in more detail the relationship between the rate of information transmission and the rate of robot motion in terms of the number of robots and the exact value of $f_{2-\text{conn}}$. Finally, we would like to explore the combination of the proposed approach with algorithms for deployment and exploration.

Bibliography

[1] Burak Aksak, Preethi Srinivas Bhat, Jason D. Campbell, Michael De Rosa, Stanislav Funiak, Phillip B. Gibbons, Seth Copen Goldstein, Carlos Guestrin, Ashish Gupta, Casey Helfrich, James F. Hoburg, Brian Kirby, James Kuffner, Peter Lee, Todd C. Mowry, Padmanabhan Pillai, Ramprasad Ravichandran, Benjamin D. Rister, Srinivasan Seshan, Metin Sitti, and Haifeng Yu. Demo abstract: Claytronics—highly scalable communications, sensing, and actuation networks. In *Proceedings of the 3rd international conference on Embedded networked sensor systems (SenSys)*, page 299, 2005.

[2] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. Distributed memoryless point convergence algorithm for mobile robots with limited visibility. *IEEE Transactions on Robotics and Automation*, 15(5):818–828, 1999.

[3] Sunny Auyang. *Foundations of Complex System Theories in Economics, Evolutionary Biology and Statistical Physics*. Cambridge University Press, Cambridge, UK, 1999.

[4] F. Bauer, J. Bristow, D. Folta, K. Hartman, D. Quinn, and J. How. Satellite formation flying using an innovative autonomous control system (autocon) environment. In *AIAA Conf. on Guidance, Navigation and Control*, pages 657–666, Reston, VA, 1997.

[5] R. W. Beard, J. Lawton, and F. Y. Hadaegh. A coordination architecture for spacecraft formation control. *IEEE Transactions on Control Systems Technology*, 9(6):777–790, 2001.

[6] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.

[7] F. Bullo, J. Cortés, and S. Martínez. *Distributed Control of Robotic Networks*. Applied Mathematics Series. Princeton University Press, 2009. To appear. Electronically available at http://coordinationbook.info.

[8] G Caprari, K O Arras, and R Siegwart. The autonomous miniature robot alice: from prototypes to applications. In *Intelligent Robots and Systems, 2000*, Takamatsu, Japan, 2000.

[9] Gilles Caprari. Autonomous micro robotics : Alice.

[10] Gilles Caprari and Roland Siegwart. Autonomous micro robotics : Alice.

[11] F. H. Clarke. *Optimization and Nonsmooth Analysis*. Canadian Mathematical Society Series of Monographs and Advanced Texts. John Wiley, 1983.

[12] J. Cortés, S. Martínez, and F. Bullo. Spatially-distributed coverage optimization and control with limited-range interactions. *ESAIM. Control, Optimisation & Calculus of Variations*, 11(4):691–719, 2005.

[13] J. Cortés, S. Martínez, and F. Bullo. Robust rendezvous for mobile autonomous agents via proximity graphs in arbitrary dimensions. *IEEE Transactions on Automatic Control*, 51(8):1289–1298, 2006.

[14] J. Cortés, S. Martínez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. *IEEE Transactions on Robotics and Automation*, 20(2):243–255, 2004.

[15] A. Das and R. Cobb. Techsat 21 - space missions using collaborating constellations of satellites. In *AIAA Conf. on Small Satellites*, Logan, UT, 1998.

[16] M. C. de Gennaro and A. Jadbabaie. Decentralized control of connectivity for multi-agent systems. In *IEEE Conf. on Decision and Control*, pages 3628–3633, San Diego, CA, December 2006.

[17] J. A. Dooley and P. R. Lawson. Technology plan for the terrestrial planet finder coronagraph. Jet Propulsion Laboratory Publication 05-8, March 2005.

[18] M. Fiedler. Algebraic connectivity of graphs. *Czech. Math. J.*, 23:298–305, 1973.

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

[20] H. Hemmati, W. Farr, B. Liu, J. Kovalik, M. Wright, and J. Neal. Formation metrology. High-level description of sensors developed at JPL for precision gormation control, November 2003.

[21] J. How, R. Twiggs, D. Weidow, K. Hartman, and F. Bauer. Orion - a low cost demonstration of formation flying in space using GPS. In *AIAA/AAS Astrodynamics Specialist Conf. and Exhibit*, pages 276–286, Reston, VA, 1998.

[22] I. I. Hussein. *Motion planning for multi-spacecraft interferometric imaging systems*. PhD thesis, University of Michigan, 2005.

[23] 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.

[24] J. W. Jaromczyk and G. T. Toussaint. Relative neighborhood graphs and their relatives. *Proceedings of the IEEE*, 80(9):1502–1517, 1992.

[25] V. Kapila, A. G. Sparks, J. M. Buffington, and Q. Yan. Spacecraft formation flying: dynamics and control. *AIAA Journal of Guidance, Control, and Dynamics*, 23:561–564, 2000.

[26] Y. Kim and M. Mesbahi. On maximizing the second smallest eigenvalue of a state-dependent graph Laplacian. In *American Control Conference*, pages 99–103, Portland, OR, 2005.

[27] P. R. Lawson and J. A. Dooley. Technology plan for the terrestrial planet finder interferometer. Jet Propulsion Laboratory Publication 05-5, June 2005.

[28] P. R. Lawson, S. C. Unwin, and C. A. Beichman. Precursor science for the terrestrial planet finder. Jet Propulsion Laboratory Publication 04-14, October 2004.

[29] H. Levine, E. Ben-Jacob, I. Cohen, and W. Rappel. Swarming patterns in microorganisms: Some new modeling results. In *IEEE Conf. on Decision and Control*, San Diego, CA, 2006.

[30] A. S. Lewis. Nonsmooth analysis of eigenvalues. *Mathematical Programming*, 84:1–24, 1999.

[31] X.-Y. Li, P.-J. Wan, and O. Frieder. Coverage in wireless ad-hoc networks. *IEEE Transactions on Computers*, 52(6):753–763, 2003.

[32] K. M. Lynch, I. B. Schwartz, P. Yang, and R. A. Freeman. Decentralized environmental modeling by mobile sensor networks. *IEEE Transactions on Robotics*, 24(3):710–724, 2008.

[33] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1997.

[34] J. E. Marsden and T. S. Ratiu. *Introduction to Mechanics and Symmetry*. Springer, 2 edition, 1999.

[35] J. A. Marshall, M. E. Broucke, and B. A. Francis. Formations of vehicles in cyclic pursuit. *IEEE Transactions on Automatic Control*, 49(11):1963–1974, 2004.

[36] S. Martínez, F. Bullo, J. Cortés, and E. Frazzoli. On synchronous robotic networks – Part II: Time complexity of rendezvous and deployment algorithms. In *IEEE Conf. on Decision and Control*, pages 8313–8318, Seville, Spain, December 2005.

[37] S. Martínez, F. Bullo, J. Cortés, and E. Frazzoli. On synchronous robotic networks - Part I: models, tasks, and complexity. *IEEE Transactions on Automatic Control*, 52(12):2199–2213, 2007.

[38] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. B. Srivastava. Coverage problems in wireless ad-hoc sensor networks. In *IEEE Conf. on Computer Communications (INFOCOM)*, volume 3, pages 1380–1387, 2001.

[39] M. Mesbahi and F. Y. Hadaegh. Formation flying control of multiple spacecraft via graphs, matrix inequalities, and switching. *AIAA Journal of Guidance, Control, and Dynamics*, 24(2):369–377, 2001.

[40] NASA. Terrestrial planet finder images. NASA Web Site.

[41] G. Notarstefano, K. Savla, F. Bullo, and A. Jadbabaie. Maintaining limited-range connectivity among second-order agents. In *American Control Conference*, pages 2124–2129, Minneapolis, MN, June 2006.

[42] 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, 2007.

[43] R. Olfati-Saber and R. M. Murray. Consensus protocols for networks of dynamic agents. In *American Control Conference*, pages 951–956, Denver, CO, June 2003.

[44] D. Peleg. *Distributed Computing. A Locality-Sensitive Approach.* Monographs on Discrete Mathematics and Applications. SIAM, 2000.

[45] S. R. Ploen, D. P. Scharf, F. Y. Hadaegh, and M. Bikdash. Initialization of distributed spacecraft formations. *Journal of the Astronautical Sciences*, 52(4), 2004.

[46] D. O. Popa, K. Sreenath, and F. L. Lewis. Robotic deployment for environmental sampling applications. In *International Conference on Control and Automation*, pages 197–202, Budapest, Hungary, June 2005.

[47] Anies Hannawati Purnamadjaja and R. Andrew Russell. Pheromone communication in a robot swarm: necrophoric bee behaviour and its replication. *Robotica*, 23(06):731–742, 2005.

[48] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.

[49] R. A. Russell. Tracking chemical plumes in constrained environments. *Robotica*, 19:451–458, 2001.

[50] P. Santi. Topology control in wireless ad hoc and sensor networks. *ACM Computing Surveys*, 37:164–194, 2005.

[51] D. P. Scharf, F. Y. Hadaegh, and S. R. Ploen. A survey of spacecraft formation flying guidance and control (part ii): control. In *American Control Conference*, pages 2976–2985, Boston, MA, June 2004.

[52] A. C. Schultz and L. E. Parker, editors. *Multi-Robot Systems: From Swarms to Intelligent Automata*. Kluwer Academic Publishers, 2002. Proceedings from the 2002 NRL Workshop on Multi-Robot Systems.

[53] M. D. Schuresko. CCLsim. a simulation environment for robotic networks, 2008. Electronically available at http://www.soe.ucsc.edu/~mds/cclsim.

[54] M. D. Schuresko and J. Cortés. Correctness analysis and optimality bounds of multi-spacecraft formation initialization algorithms. In *IEEE Conf. on Decision and Control*, pages 5974–5979, San Diego, CA, December 2006.

[55] M. D. Schuresko and J. Cortés. Safe graph rearrangements for distributed connectivity of robotic networks. In *IEEE Conf. on Decision and Control*, pages 4602–4607, New Orleans, LA, 2007.

[56] M. D. Schuresko and J. Cortés. Distributed motion constraints for algebraic connectivity of robotic networks. In *IEEE Conf. on Decision and Control*, pages 5482–5487, Cancun, Mexico, December 2008.

[57] M. D. Schuresko and J. Cortés. Distributed motion constraints for algebraic connectivity of robotic networks. In *Journal of Intelligent and Robotic Systems*, 2009. To appear.

[58] M. D. Schuresko and J. Cortés. Distributed tree rearrangements for reachability and robust connectivity. In *International Conference on Hybrid Systems: Computation and Control*, San Francisco, CA, 2009. To appear.

[59] M. D. Schuresko and J. Cortés. Distributed tree rearrangements for reachability and robust connectivity. *SIAM Journal on Control and Optimization*, 2009. Submitted.

[60] Mac Schwager, James McLurkin, and Daniela Rus. Distributed control algorithms for networked mobile robots with sensory feedback.

[61] Mac Schwager, James McLurkin, and Daniela Rus. Distributed coverage control with sensory feedback for networked robots. In *Proceedings of Robotics: Science and Systems*, Philadelphia, PA, August 2006.

[62] S. L. Smith, M. E. Broucke, and B. A. Francis. A hierarchical cyclic pursuit scheme for vehicle networks. *Automatica*, 41(6):1045–1053, 2005.

[63] D. P. Spanos, R. Olfati-Saber, and R. M. Murray. Dynamic consensus on mobile networks. In *IFAC World Congress*, Prague, Czech Republic, July 2005.

[64] M. Tillerson, G. Inalhan, and J. How. Coordination and control of distributed spacecraft systems using convex optimization techniques. *International Journal on Robust and Nonlinear Control*, 12:207–242, 2002.

[65] T. Vicsek, A. Czirók, E. Ben-Jacob, I. Cohen, and O. Shochet. Novel type of phase transition in a system of self-driven particles. *Physical Review Letters*, 75(6-7):1226–1229, 1995.

[66] C. W. Wu. Algebraic connectivity of directed graphs. *Linear and Multilinear Algebra*, 53(3):203–223, 2005.

[67] P. Yang, R. A. Freeman, G. Gordon, K. M. Lynch, S. Srinivasa, and R. Sukthankar. Decentralized estimation and control of graph connectivity for mobile sensor networks. In *American Control Conference*, Seattle, WA, 2008.

[68] M. M. Zavlanos and G. J. Pappas. Controlling connectivity of dynamic graphs. In *IEEE Conf. on Decision and Control*, pages 6388–6393, Seville, Spain, 2005.

[69] M. M. Zavlanos and G. J. Pappas. Distributed connectivity control of mobile networks. In *IEEE Conf. on Decision and Control*, New Orleans, LA, 2007.